CONSTANT VOLTAGE HOT-WIRE ANEMOMETRY FOR THE BOUNDARY LAYER DATA SYSTEM

A Thesis presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

by

Hon Yee Li

December 2013

COMMITTEE MEMBERSHIP

TITLE:	Constant Voltage Hot-Wire Anemometry for the Boundary Layer Data System
AUTHOR:	Hon Yee Li
DATE SUBMITTED:	December 2013
COMMITTEE CHAIR:	Russell V. Westphal, Ph.D Professor, Mechanical Engineering
COMMITTEE MEMBER:	Kim A. Shollenberger, Ph.D Professor, Mechanical Engineering
COMMITTEE MEMBER:	Christopher C. Pascual, Ph.D Professor, Mechanical Engineering

ABSTRACT

Constant Voltage Hot-Wire Anemometry for the Boundary Layer Data System

Hon Yee Li

To continue the development of the Boundary Layer Data System (BLDS), a constant voltage hot-wire anemometer (CVA) is implemented into the BLDS for flight-testing. The hot-wire anemometer was chosen as an alternative to the traditional pressure probe because of the ability to measure both average velocity and fluctuating velocity within the boundary layer. Previous work done on the benchtop has led to the design of miniaturization, flight-capable hardware for the BLDS. The next step in the development of the BLDS – CVA calls for quantifying the accuracy of the boundary layer measurements measured by the CVA system. To do this, numerous turbulent boundary layer velocity and fluctuating velocity profiles were taken on a flat-plate at various speeds within the Cal Poly 2x2 wind tunnel with both the traditional pressure probe and the CVA. These test resulted showed agreement between the hot-wire and pressure probe data. Once this was completed the new CVA hardware was tested along with the new software that was written for the BLDS – CVA. In addition, due to the limited memory space onboard the BLDS – CVA, an approximation had to be developed to convert the average voltage data from the BLDS – CVA to the average velocity data due to the non-linear calibration function. The approximation developed was able to match the exact values from a traditional calibration. Lastly, due to the inability to perform a laboratory calibration of the hot-wire at altitude, where the conditions differ significantly from the ground conditions, a new procedure for hot-wire calibration was developed. The method developed was validated through wind tunnel testing and a computer thermal/electric model. With the completion of this work, the BLDS – CVA is ready for flight-testing.

Keywords: Hot-wire, constant voltage anemometer, laminar, turbulent, boundary layer, BLDS, calibration, temperature drift correction

ACKNOWLEDGMENTS

I would like to thank Russell Westphal for all the knowledge he was able to pass on to me during the 1.5+ years I have worked on the BLDS project and this great opportunity that he has given me. I would also like to thank him ahead of time for any career opportunities I will obtain in the future because I know the building blocks I gained from working with Dr. Westphal allowed me to be where I will be.

I would like to thank Kim Shollenberger and Christopher Pascual for serving on my committee and their invaluable advice they have contributed towards my work.

I would like to thank Don Frame for his contribution to the electronics of the BLDS program. Without his work, I would not be able to complete any of the work I have done for this thesis.

I would like to thank Chris Harris, Anne Bender, and the people at Northrop Grumman Corp. for their irreplaceable knowledge that they contributed towards the hot-wire work and their financial support for the BLDS program.

Finally I would like to thank my family, friends, co-workers, classmates, professors, teachers, coaches, and anyone else I missed for all their endless support, advice, and help they have given me. Without them, I would not be where I am today.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

C_f	=	Local coefficient of skin friction
d	=	Outer diameter of pressure probe, inches
D	=	Diameter of wire on the hot-wire probe, μm
E	=	Mean voltage output from HWA, V
E_∞	=	Mean voltage output from HWA in the freestream, V
I_w	=	Hot-wire probe current, V or mA
k	=	Exponent for inverted calibration curve power law
k_f	=	Thermal conductivity of air at film temperature, W/m-K
Nu	=	Nusselt Number
OHR	=	Traditional overheat ratio, ratio of hot to cold probe resistance
P	=	Constant offset coefficient for inverted calibration curve power law, kPa or mm Hg
P_∞	=	Ambient pressure, kPa or mm Hg
Q	=	Multiplier coefficient for inverted calibration curve power law
Re	=	Reynolds number
Re_f	=	Reynolds number of film
R_1	=	HWA fixed circuit resistor, 5000 Ω in CVA prediction model
R_2	=	HWA fixed circuit resistor, 50 Ω in CVA prediction model
R_F	=	CVA circuit resistor, 1000 Ω in CVA prediction model
R_L	=	Resistance of probe connections in anemometer circuit, Ω
R_w	=	Resistance (hot) of hot-wire probe at operating temperature, Ω
R_∞	=	Resistance (cold) of hot-wire probe at ambient temperature, Ω

T_f = Film Temperature, ℃ or K

T_∞ = Ambient fluid temperature, ℃ or K

u = Mean velocity, m/s

u' = Mean fluctuating velocity, m/s

u_τ = Shear velocity, m/s

U_∞ = Freestream velocity, m/s

V_1 = Voltage source powering HWA circuit, V

V_o = Op-amp output from prototype CVA system, V

V_p = Voltage output from the Pitot tube differential pressure transducer, V

V_S = Op-amp output voltage for simplified circuit in literature, V

V_w = Hot-wire probe voltage set-point, V

$V_{w,in}$ = Voltage input by BLDS – CVA circuitry to the hot-wire probe, V

$V_{w,out}$ = Voltage observed across the hot-wire probe on BLDS – CVA, V

x = Distance from where the boundary layer starts forming, inches

y = Distance perpendicular to the surface, inches

z = Variables contributing to the measurement uncertainty analysis

δ = Boundary layer thickness, inches

δ^* = Displacement thickness, inches

μ_f = Air viscosity at film, N-s/m^2

ρ_f = Air density of film, kg/m^3

θ = Momentum thickness, inches

1. INTRODUCTION

The purpose of this thesis is to develop a version of the Boundary Layer Data System (BLDS) that incorporates a constant-voltage hot-wire anemometer. The addition of hot-wire anemometry would expand the capability of the BLDS by adding the ability to measure both average velocities and velocity fluctuations with a single device. The addition of a hot-wire anemometer to the BLDS builds on previous work at Cal Poly which demonstrated the potential of the Constant Voltage Anemometer (CVA) to satisfy the requirements for a flight-capable hot-wire approach needed for BLDS. The challenges for the current work include:

- Evaluating the accuracy of CVA measurements of boundary layer mean and fluctuating velocity profiles
- Testing of the new, miniaturized hardware required by the BLDS – CVA and developing the new software to be required for the BLDS – CVA
- Answering the question of "How to compute the boundary layer data using the averaged measured values of input current saved by the BLDS instead of the 'raw' values of input current measured from the hot-wire anemometer?" due to a limitation in data capacity of the BLDS
- Solving the in flight hot-wire anemometer calibration problem.

The big picture goal of this project is to use BLDS to make measurements of boundary layer characteristics in flight to support the development of aircrafts, such as High Altitude Long Endurance (HALE) aircraft, that have substantial region of laminar flow over the wing. Laminar flow is important in these aircrafts because it minimizes the drag force through the reduction of skin friction drag. This allows for increase fuel

efficiency and therefore save fuel or increase the duration of HALE aircraft's flights. Several hours can be added to the flight duration of an aircraft by having laminar flow present over a significant portion of the wing instead of having turbulent flow. This is simply because laminar flow has significantly less skin friction drag than turbulent flow. When comparing the skin friction drag of laminar flow to that of turbulent flow, the skin friction drag of laminar flow is 10 – 20% of that of turbulent flow. With skin friction drag accounting for approximately half of the total drag of an aircraft, and assuming half of that is contributed by the wing, the drag due to having laminar flow over a wing can decrease the amount of total drag of an aircraft by 10 - 15% [1].

Due to this fact, many engineers use theory-based analysis, numerical models, and wind tunnel testing in order to ensure that the aircraft of interest does achieve laminar flow where practical. However, computational models seldom predict an accurate transition point between laminar and turbulent flow and data from wind tunnel testing is very sensitive to the testing conditions. In addition, it is very difficult to simulate environmental test conditions experienced in flight in a wind tunnel environment. As a result, there is no guarantee that the aircraft will achieve laminar flow where expected and the performance benefits due to laminar flow cannot be credited for unless it is observed directly during the aircraft's normal in flight operation. To confirm that laminar flow is observed over the wing where expected, a device known as the BLDS can be implemented during test flights.

The BLDS is one of two devices that have been developed and improved on since 2005 to measure boundary layer characteristics on the surface of aircrafts or flight-borne models [2]. The second of the two devices is the Preston Tube Data System (PTDS). The

PTDS solely measures boundary layer data right on the surface using fixed-position probes, whereas the BLDS incorporates a stage that can move a probe within the boundary layer [3]. Both of these devices are flight ready, self-contained, and do not require mechanical modification of the aircraft or model's surface. The current version of the PTDS measures the boundary layer data by using three pressure probes. One is a surface static pressure probe known as a Sproston-Goskel probe [4], the second is a freestream total pressure probe, and the third is a surface total pressure probe known as a Preston tube [5]. With the use of the three probes mentioned above, plus temperature measured with its on-board sensor, the local skin friction and the corresponding drag force due to skin friction can be computed. Using this information, it can be determined if the flow is laminar, turbulent, or in the transition region. In addition, the flow temperature can be measured with the onboard temperature sensor and the local pressure can be calculated using a single point calibration in conjunction with the pressure data measured by the Sproston-Goskel probe. The BLDS, on the other hand, differs from the PTDS because of its ability to capture a complete boundary layer velocity profile. With a complete boundary layer velocity profile, the boundary layer thickness, displacement thickness, and momentum thickness can be computed in addition to the local skin friction coefficient and drag force due to skin friction. This is achieved by attaching a probe onto a stage that allows for the probe to move throughout the boundary layer and the flow. In the current version of BLDS [6], the probe on the stage can be a total pressure probe, a Conrad probe [7], or a rotatable probe [8]. As with the PTDS, the BLDS also has a temperature sensor that can measure the local flow temperature and the local static and

freestream pressures are obtained using the same fixed probes and sensors as used by the PTDS.

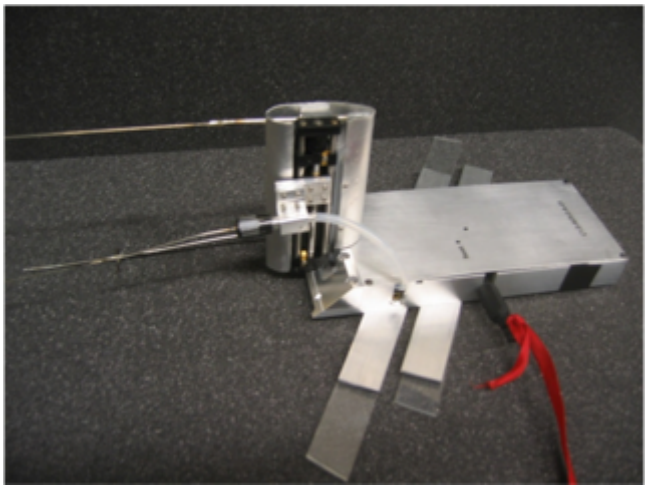


Figure 1.1. Boundary Layer Data System (BLDS) with total pressure probe.

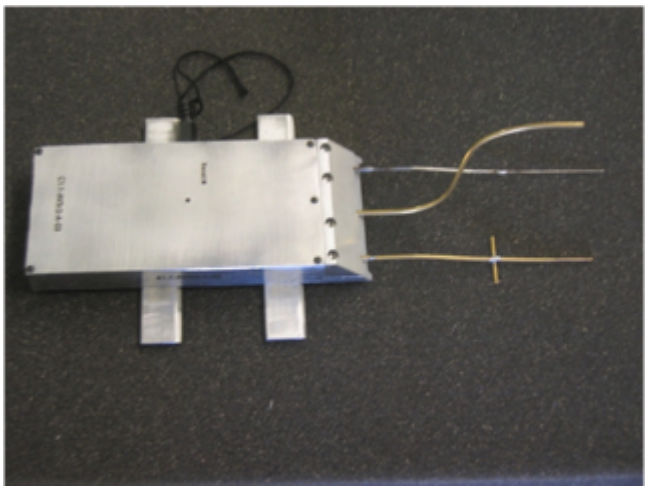


Figure 1.2. Preston Tube Data System (PTDS).

Although the current BLDS version has the capabilities to adopt different probes as mentioned above, the addition of hot-wire anemometry for BLDS would give the additional capability of measuring velocity fluctuations. This extra bit of information would quantify the turbulence intensities present within the boundary layer, allowing for explicit determination of the laminar/turbulent state of the boundary layer. To measure

velocity fluctuations accurately in all possible scenarios in a boundary layer, the probe used to measure velocity fluctuations has to have a frequency response as high as the velocity fluctuations seen in turbulent boundary layers. This is due to turbulent boundary layers having the highest velocity fluctuation frequency. A study done by Wazzan [9] with the Blasius flat plate boundary layer profile has shown that velocity fluctuation in a turbulent boundary layer can fluctuate up to a frequency of 5,720 Hz for a freestream velocity of $U_\infty = 50\, m/s$ at a distance $x = 1.1\, inches$ from where the turbulent boundary layer starts forming. At a distance further downstream, at $x = 10\, inches$, the velocity fluctuation frequency drops off significantly to around 2,000 Hz. These characteristics of a turbulent boundary layer makes hot-wire anemometry a primary choice for the BLDS to achieve the ability to measure velocity fluctuations in a boundary layer. Hot-wires anemometers have the ability to capture these fluctuations as they have a frequency response of up to 100,000 Hz. However, the actual frequency response of the hot-wire varies depending on the type of system used to operate the hot-wire and there are three systems to choose from.

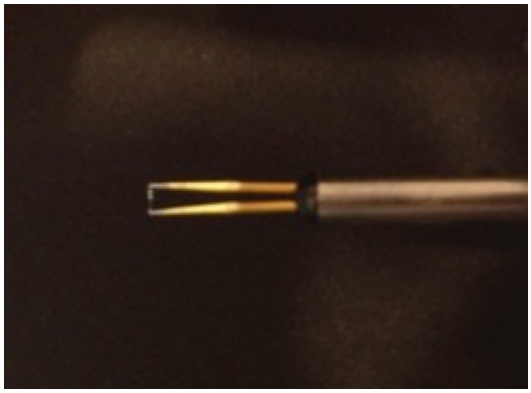


Figure 1.3. Preston Tube Data System (PTDS).

The three types of hot-wire anemometers systems are the Constant Current Anemometer (CCA), Constant Voltage Anemometer (CVA), and the Constant

Temperature Anemometer (CTA). In the CCA, the circuit is developed to keep the current constant across the hot-wire. This is necessary because the resistance of the hot-wire changes as the temperature of the wire changes. The idea of keeping the current across the hot-wire constant in the CCA is the similar for the CVA and the CTA. However, instead of keeping the current constant, the CVA keeps the voltage across the hot-wire constant and the CTA keeps the temperature of the hot-wire constant.

In order to evaluate which of these three systems is the best solution for the BLDS, a comprehensive study was done previously on the benchtop by Will Neumeister [10]. Table 1.1 provides a summary of the characteristics of each system for a sample measurement situation.

Table 1.1. Comparison summary between CCA, CTA, and CVA. Parameters quoted for 3.8 micron diameter, platinum coated tungsten hot-wire probe at STP conditions, U = 50 m/s and OHR = 1.8 [10].

	Time Constant (ms)	Sensitivity (mV/(m/s))	Half-Amp. (-3 dB) Cut-Off Frequency	Set-Up & Tuning
CCA	≈ 0.33	≈ 15	≈ 800 Hz	Set Current Level, Tune Bridge
CVA	≈ 0.13	≈ 6	≈ 2200 Hz	Set Voltage Level
CTA	< 0.05	≈ 9	≈ 100 kHz	Set Overheat, Tune Bridge, Square Wave Test

The CCA method was found to provide a simple electrical circuit and has the highest sensitivity of all three methods at $15\frac{mV}{m/s}$. However, due to the nature of the circuitry, it is prone to hot-wire burnout and requires a manual tuning of the bridge in the electrical circuit to achieve best frequency response. This poses a huge problem during the operation of the BLDS which is a self-contained device, autonomous device so it is impractical to tune the bridge on the CCA circuit during the flight. Furthermore, the

frequency response of the CCA is poor: attenuation of frequencies above 800 Hz for the sample measurement situation would not provide sufficient frequency response to measure accurate velocity fluctuation data.

The CTA method provides the highest frequency response of up to 100,000 Hz and has no risk of wire burnout. However, the CTA requires a manual setting of the overheat ratio (OHR) and careful tuning of the bridge using a square wave test signal to achieve stable operation and best frequency response. This requirement for manual tuning makes CTA impractical for use with BLDS.

Lastly, the CVA method provides better frequency response than CCA and has the key advantage that it requires no manual tuning of its constant-voltage control circuitry -- it only requires the setting of the wire voltage. However this method requires the user to ensure that the voltage setting is not high enough to cause hot-wire burn-out. In addition, an increase of flow speed causes the OHR to decrease and this lowers both the frequency response and sensitivity of the hot-wire. Notwithstanding its disadvantages, it was found that the CVA was the best solution for the BLDS because of the performance characteristics of the CVA and the ability to develop solutions to reduce the effects of the disadvantages explained in this paragraph. The solutions to these disadvantages will be addressed in later chapters of this thesis. Conversely, the disadvantages from using the CCA or the CTA are difficult to overcome and not practical due to the solution defeating the purpose of the BLDS being a self-contained device.

The CVA is a system developed by Sarma in 1990 [11] and the circuit design can be seen below in Figure 1.4. Where R_L represents any lead resistance used to connect the hot-wire, R_w, to the CVA circuit. In most cases R_L is significantly smaller than R_w. The

way the CVA circuit works is that the user specifies the desired voltage, V_w, across the hot-wire, through selection of resistance values for R_1 and R_F in the CVA circuit and the voltage source, V_1. The relationship between these values can be seen in equation (1.1) here and derived as seen in [12], [13]

$$V_w = \frac{R_F}{R_1} V_1. \tag{1.1}$$

As flow is introduced around the hot-wire, the hot-wire resistance decreases dramatically due to the increase in heat transfer coefficient as the main heat transfer mode switching from natural convection to forced convection. The drop in resistance for a constant current, results in the voltage across the hot-wire to drop. As the voltage in the hot-wire drops, the op-amp in the circuit signals the voltage source to allow more current to flow so it compensates for the drop in resistance and brings the voltage across the hot-wire back to the specified value. This can be seen through Ohm's law

$$V_w = I_w(R_w + R_L). \tag{1.2}$$

With the way the circuit naturally compensates for the change in resistance of the hot-wire, it helps with wire burnout by decreasing the voltage output in the circuit if the velocity of the flow drops around the hot-wire. However, the voltage output will only drop back down to the minimum voltage needed to keep the wire voltage constant with no flow. So careful calculation in the choice of the wire voltage, V_w, should be taken before operating the CVA in different ambient conditions. In addition, the nature of the circuit allows for improvement of the frequency response over the CCA, but it still cannot achieve the high frequency response seen in the CTA. Furthermore, as specified before, keeping the wire voltage constant does not keep the OHR constant, which

changes the sensitivity and the frequency response of the hot-wire throughout any measurement scenario wherein flow velocity varies.

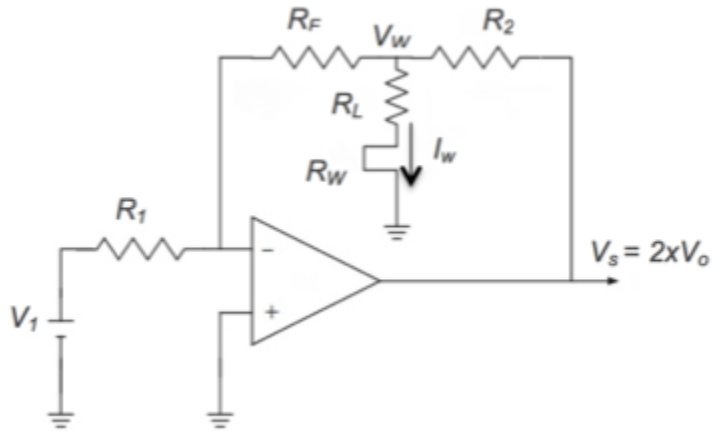


Figure 1.4. Schematic of a simplified CVA circuit.

To complete the full integration of the CVA system into the BLDS, wind tunnel experiments are to be performed along with the support of the thermal/electric model of the CVA developed by Will Neumeister [10], [14]. First, the evaluation of the accuracy of the CVA was completed using the benchtop CVA electronics. This was done by comparing the data from the CVA boundary layer velocity profile against a proven method, the total pressure (Pitot) probe, as well as published data. The results from this study will be presented in the following chapter. In the succeeding chapter, the changes made to the BLDS hardware and software will be thoroughly explained along with the results from testing the hardware and software. This includes the operation of the miniaturized CVA circuit for BLDS, called the CVA Daughter Board, the software changes needed to allow for the CVA operation, measurement and storage of the velocity fluctuation data, and the hardware changes needed to ensure accurate measurement of the fluctuating velocity flow data. In addition, the mathematical formulation developed to address the limited memory space on the BLDS – CVA will be presented. This method

was verified by comparing the boundary layer data results from using the “raw data” and the boundary layer data results from using the averaged data with the mathematical formulation. In the final chapter, the problem of calibrating a hot-wire for in flight conditions will be presented and addressed. The solution to this problem was verified by comparing boundary layer data results from various wind tunnel tests and theoretically with the thermal/electric CVA model.

2. BENCHTOP CVA MEASUREMENTS IN A BOUNDARY LAYER

2.1. Experimental Setup

In order to evaluate if the CVA for its intended application for laminar/turbulent boundary layer, its measurements were compared to those from a traditional total pressure probe within the turbulent boundary layer on the surface of a flat plate. For this evaluation, the benchtop CVA electronics were used, with the CVA output read by a laptop computer through a USB-connected analog-to-digital converter. The test was conducted in Cal Poly's Mechanical Engineering Fluids Lab using the 2-foot by 2-foot wind tunnel. The wind tunnel has the ability to achieve a maximum velocity of 110 mph, or $50\ m/s$, but the boundary layer profile data will be measured in the $U_\infty = 22 - 47\ m/s$ range. The data was taken with the following test setup: a 0.02 inch diameter trip wire spanning the width of the 3 foot long, 2 foot wide flat plate 3 inches downstream of the leading edge, the pressure probe and hot-wire located 28 inches behind the leading edge, and a Pitot-static probe located 6 inches from the ceiling of the test section with the static port lined up with the pressure probe and hot-wire sensor. In addition, the flat plate has four washers on each of the rearmost supports to tilt the flat plate nose down at an angle of 0.58 degrees and a flap on the trailing edge. These two features are present on the flat plate in the test setup to create a slight favorable pressure gradient in the test section to ensure attached laminar flow at the sharp leading edge of the flat plate. The test setup for the CVA can be seen in Figure 2.1.1 - Figure 2.1.3 below.

The pressure probe experiment has the exact same setup, however, instead of the hot-wire being mounted to the stage, the pressure probe takes the place of the hot wire.

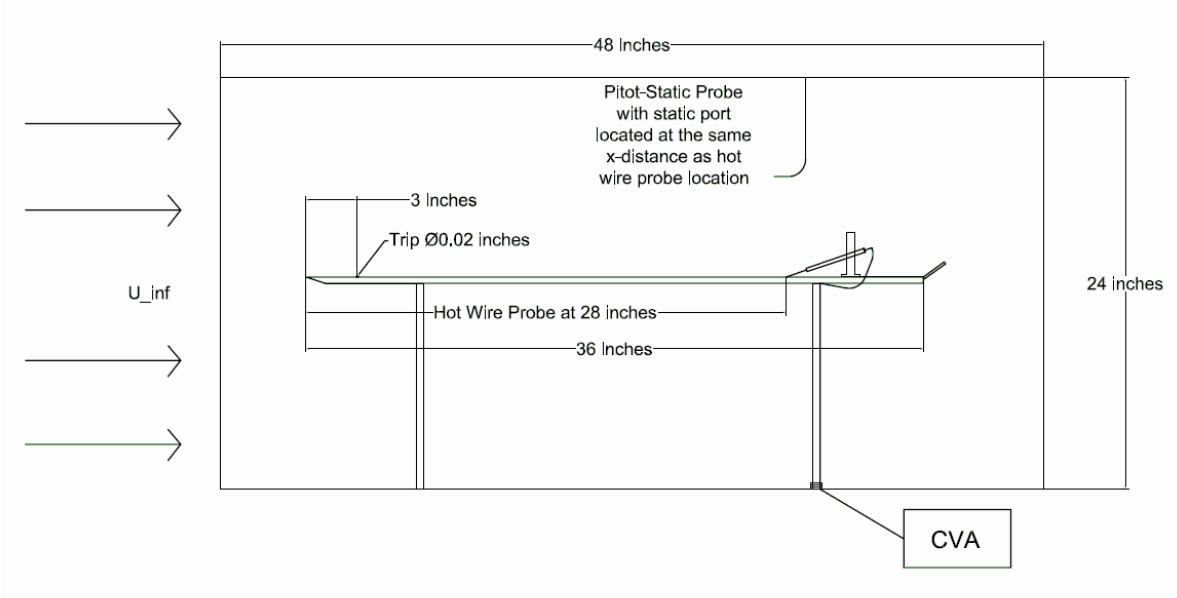


Figure 2.1.1. Schematic of CVA experimental setup used to collect a turbulent boundary layer profile. (Note: Figure not to scale)

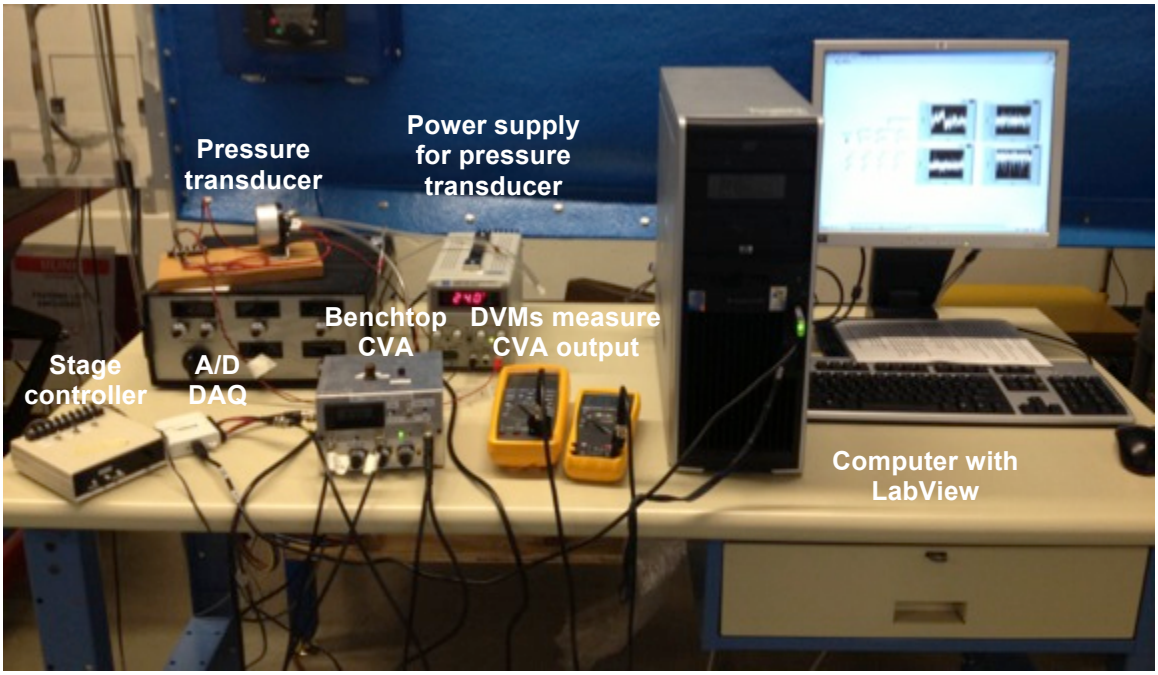


Figure 2.1.2. Electronics setup for the CVA benchtop turbulent boundary layer tests.

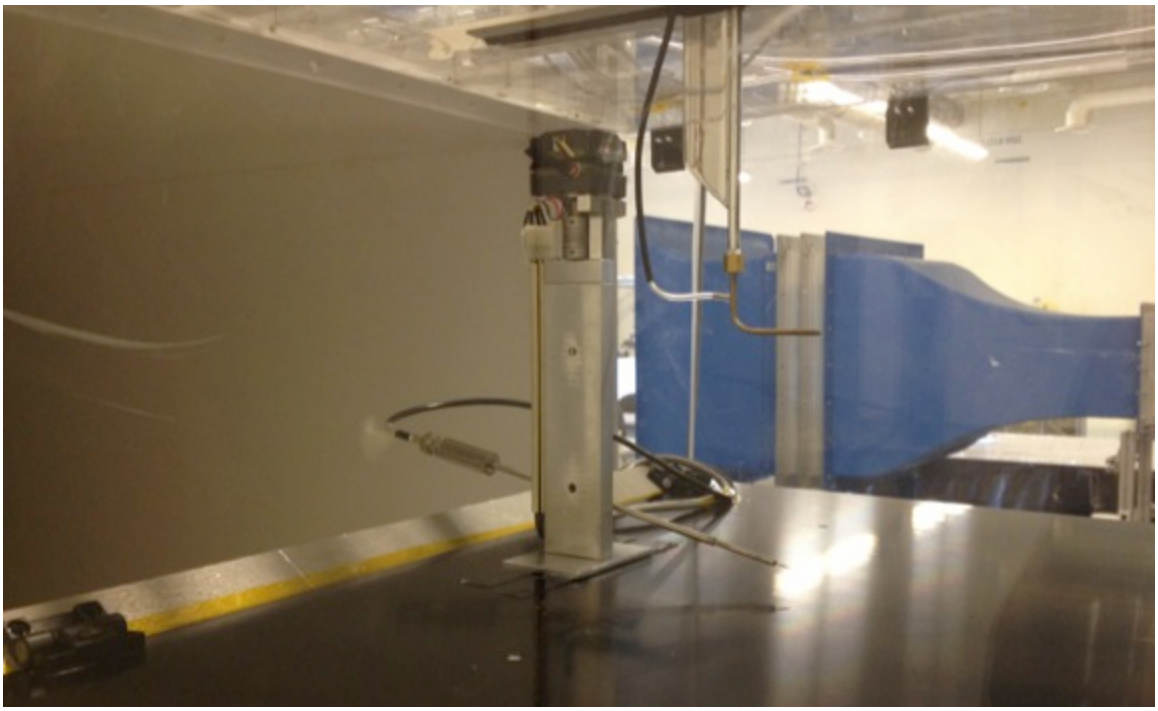


Figure 2.1.3. Motorized stage, hot-wire probe, and Pitot-static probe setup for benchtop CVA turbulent boundary layer tests.

The data acquisition program used in this experiment is National Instruments LabView and the data acquisition device used is the NI USB-6009, connected to a laptop computer using a USB interface. The program written for collecting the CVA voltage outputs in LabView was originally created by Will Neumeister [10] and improved on by the author. For the experiments shown in this chapter, the program was instructed to take 1,000 data points for 5 seconds at each location point. Then the program sends a command to the stepper motor to move a user-specified 40 encoder counts (0.01575 inches) upwards from the previous position. This is done for a user specified 50 positions. The program also waits for a conservative, user specified, 5 seconds in between positions to allow the hot-wire to adjust to the new flow conditions. The program's user interface can be seen below in Figure 2.1.4. The pressure probe used had an OD of $d = 0.0325\ inches$ and the TSI 1210-T1.5 hot-wire was used. The TSI 1210-T1.5 hot-

wire has a diameter of $D = 3.8\,\mu m$ and a cold resistance of $R_\infty = 6.05\,\Omega$ cold at $T_\infty = 21.4$℃. The pressure sensor used to measure the differential pressure between the total pressure and static pressure was the Setra 239 digital pressure transducer and the voltage output was recorded by the NI USB-6009 which provided an A/D range of $0 - 5V$ and 14 bits of resolution. The CVA was set for $V_w = 0.69V$ and the R_w and I_w was outputted to Fluke 179 and Fluke 289 respectively for manual recording and monitoring during the calibration process. The V_o output is connected from the CVA to the NI USB-6009 for data recording. Ambient conditions were measured using a Paroscientific, Inc. Digiquartz Laboratory Standard Model 745.

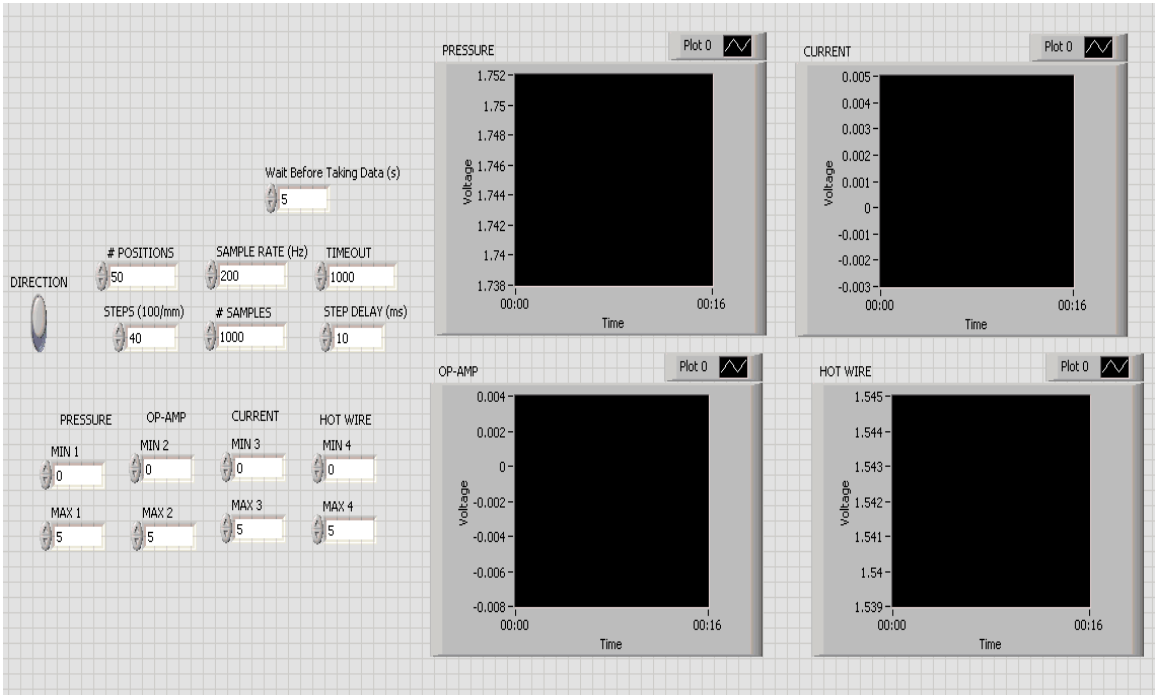


Figure 2.1.4. User interface of the LabView program created to capture the boundary layer profile data.

2.2. Experimental Test Results

With the experiment setup described above, four tests were ran at $U_\infty = 22.4\,m/s, 30.3\,m/s, 37.9\,m/s$ and $44.8\,m/s$ for the hot-wire and four more test were

ran at $U_\infty = 22.6\ m/s, 30.2\ m/s, 38.3\ m/s$ and $45.9\ m/s$. The processed data collected from the experiment can be seen in Figure 2.2.1 and Figure 2.2.2. Note that a traditional laboratory hot-wire calibration was performed to assist with computing the CVA data output seen throughout this chapter [10].

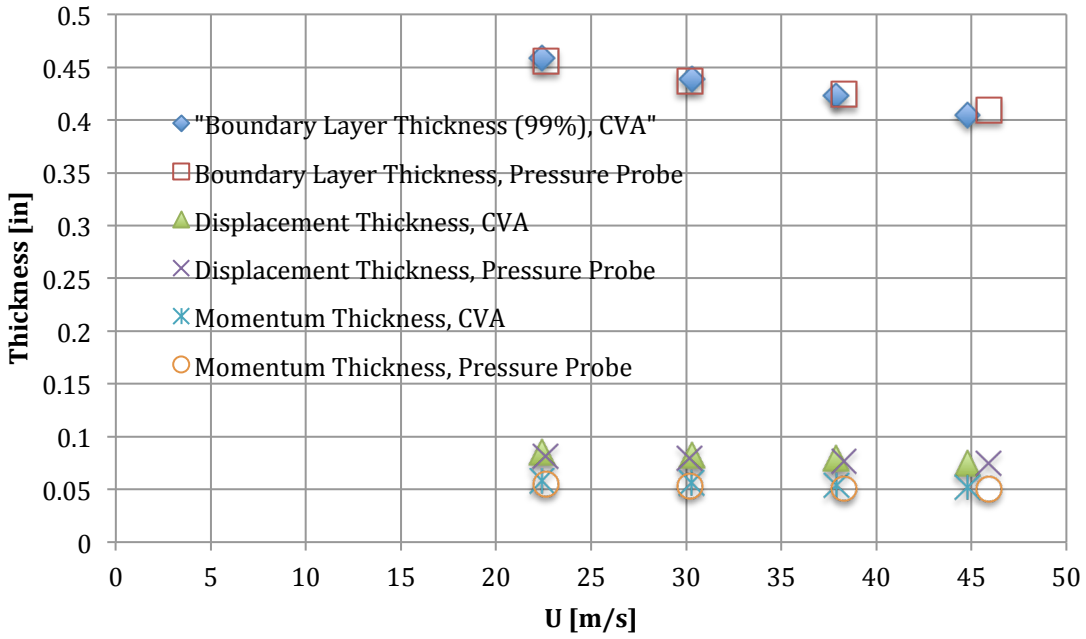


Figure 2.2.1. Turbulent boundary layer thickness characteristics across the four freestream velocities tested. Note: Data was not corrected for temperature drift.

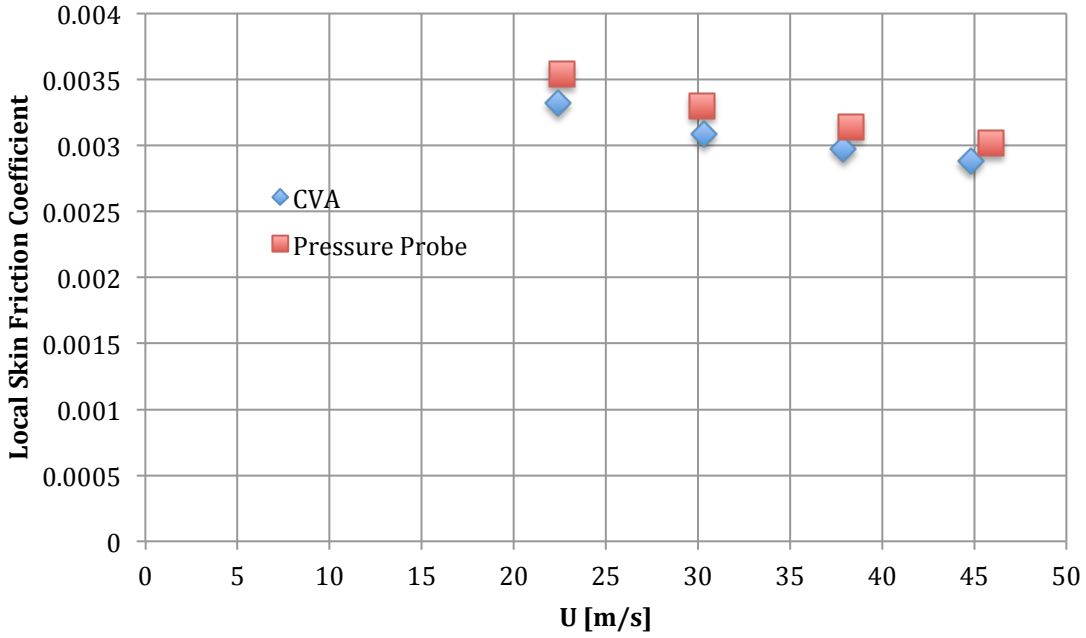


Figure 2.2.2. Local skin friction coefficient values across the four freestream velocities tested. Note: The data was not corrected for temperature drift.

From the processed data seen in Figure 2.2.1 and Figure 2.2.2, it can be concluded that the CVA can measure boundary layer data as accurate as a pressure probe. When comparing the CVA data to the pressure probe data, the percent error ranged anywhere from 0% to 6.4% for the parameters from the processed data. The error contributed by the measurement error can be seen in Appendix A. One noticeable trend from the computed parameters is that the CVA results for thickness parameters are mostly higher than those from the pressure probe data, whereas the skin friction is mostly lower from the CVA.

One source of uncertainty in these results concerns the determination of the starting position above the surface for the hot-wire. In the lab where these experiments were performed, more sophisticated equipment necessary to determine the starting position of the hot-wire, without the risk of damaging the hot-wire, was not readily available. Therefore the starting position of the hot-wire was determined from visual inspection. Using visual inspection, it was determined that the closest the hot-wire can

start from the flat plate without using additional instrumentation is 0.01 inches and can be no worse than 0.04 inches away from the flat plate. Given this assumption, it is assumed that it was possible to get the hot-wire starting position as close as was deemed possible, which is 0.01 inches from the flat plat with an estimated uncertainty of $\pm$ 0.01 inches.

The discrepancy between the hot-wire and pressure probe boundary velocity profiles is mainly due to the flow temperature drift which occurred while collecting the data. With the test setup described in Section 2.1, the time it takes to run the calibration of the hot-wire and the four experiments exceeds an hour. During this time, the wind tunnel test section temperature rises an estimated 3℃ when the ambient temperature at the beginning of the test starts at 21.4℃, which is the ambient temperature during the calibration of the hot-wire for this experiment. This temperature drift significantly affects the data collected by the CVA. With the way the hot-wire works, if the temperature of the test conditions increases during the experiment from when the hot-wire is calibrated, the hot-wire reads a lower velocity than what it is actually seen. This is due to a reduction in heat transfer between the flow and the hot-wire and as a result a lower voltage output is required by the CVA to keep the voltage across the wire constant. This lower voltage then is associated with a lower velocity through the calibration function, see Figure 2.2.3. Section 4.2 provides a more detailed explanation of this phenomenon.

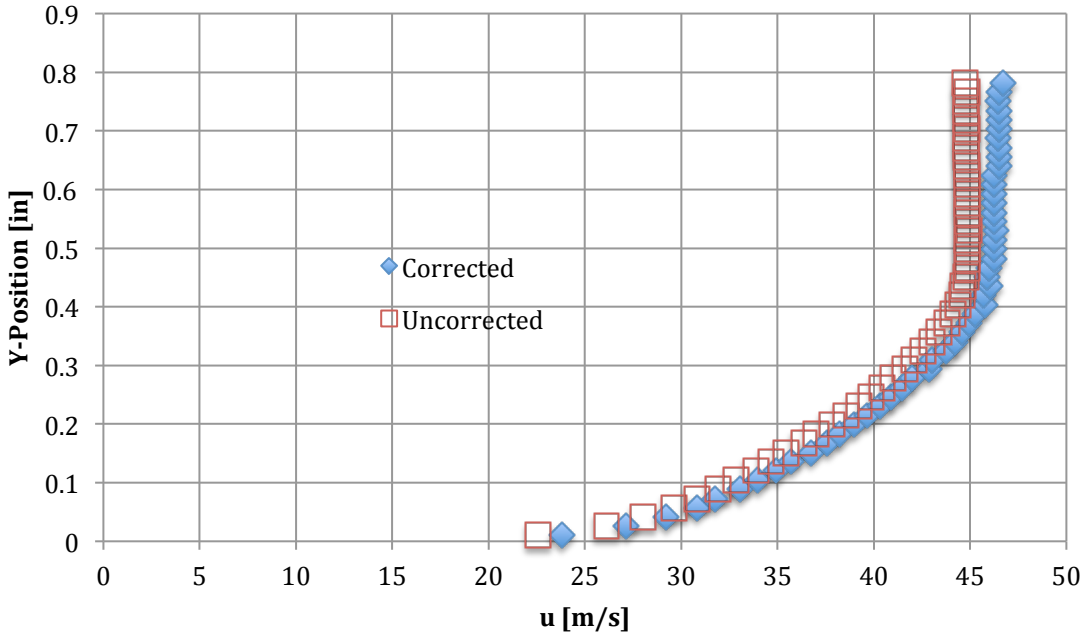


Figure 2.2.3. Comparison of the temperature corrected and uncorrected turbulent boundary layer velocity profile using the CVA for U = 44.8 m/s.

An attempt was made to correct the data for the temperature drift. First the calibration of the hot-wire was corrected for a nominal 21.4℃ as the flow temperature drift during the calibration process in the wind tunnel increased by 1.8℃. Next this drift-corrected calibration function is used to convert the voltage outputted by the CVA to get a reference velocity. Now using the reference velocity, original voltage, and the temperature drift along with the thermal/electric CVA model, the expected change in voltage due to the drift in temperature can be predicted. For example, it was found that for a $\Delta T_\infty = 2.9$℃ and a flow velocity of $U_\infty = 43\ m/s$, the wire voltage read a value that was 0.013V lower than expected resulting in a velocity change of 3.7 m/s. Note that since temperatures were measured at various times throughout the experiment, it was assumed that temperature varies linearly with experiment time. The result of this correction is seen above in Figure 2.2.3.

Although the corrected boundary layer velocity profile does not result in a smooth boundary layer velocity profile due to the assumption of temperature changing linearly throughout the experiment including the freestream points, this study shows that a correction is still necessary to address the temperature drift observed in the experimental tests. Without the correction a freestream velocity of $U_\infty = 44.8\ m/s$ would be measured by the CVA instead of the known $U_\infty \cong 47\ m/s$ that should be measured at the highest speed setting of the wind tunnel. With the correction performed in this study, the freestream velocity is $U_\infty = 46.8\ m/s$, which is very close to what is seen in the pressure probe case. The $2\ m/s$ increase in velocity due to the temperature drift is about a 4.5% increase for the given freestream velocity or 1.5% increase in velocity per degree Celsius change. This discrepancy can be observed in the uncorrected turbulent boundary layer profile seen in Figure 2.2.4 - Figure 2.2.7. Note that this correction only works well with small temperature changes as the voltage output difference due to the change in ambient temperature is not linear. This can be seen in the boundary layer velocity profiles in Section 4.5.

In Figure 2.2.4– Figure 2.2.7, the uncorrected turbulent boundary layer profile for each velocity taken by the pressure probe and CVA is plotted on the same graph. It can be seen in the $U_\infty = 22.6\ m/s$ case that the boundary layer profile captured by the CVA is shifted upwards when compared to the pressure probe. This might suggests that the starting position of the hot-wire was closer than expected. However, a linear shift of the y-location of the CVA data to match the pressure probe data exactly would require the hot-wire sensor starting position to be unrealistically close to the surface. In fact, the first point of the CVA data would be a negative distance from the plate, suggesting that the

hot-wire started inside the plate. If the y-location of the CVA data was shifted downwards to the lowest realistic number, 0 inches, the turbulent boundary layer velocity profile will still look like it is shifted upwards but the severity of the shift will not be as extreme as seen in Figure 2.2.4. Note that the temperature drift in this test does not play as large of a role as the other tests later in the experiment because only a temperature drift of a half of a degree Celsius was predicted.

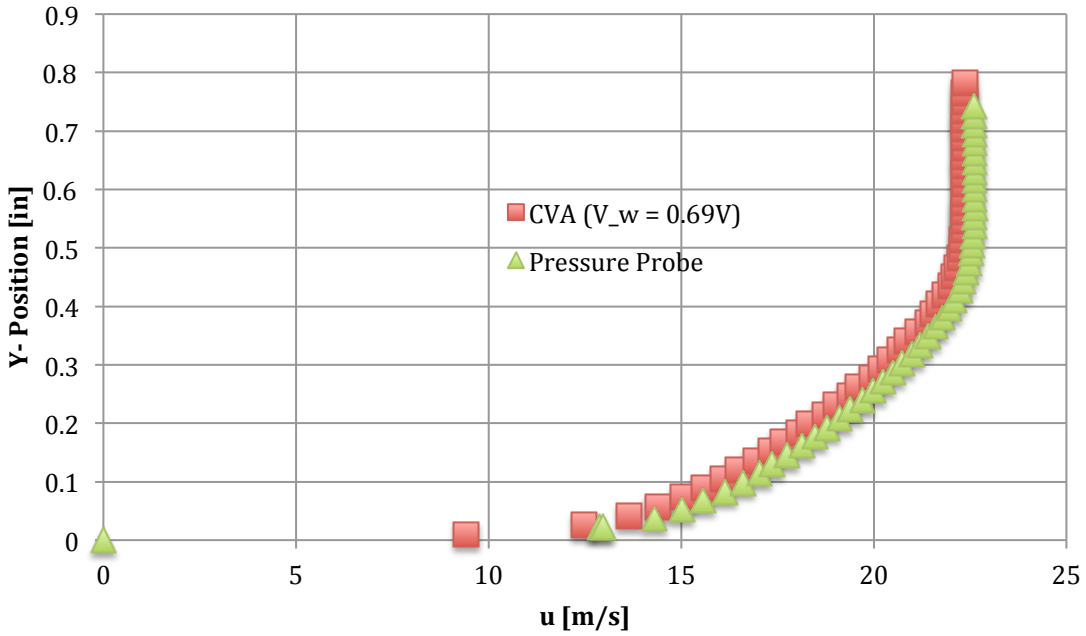


Figure 2.2.4. Turbulent boundary layer profile comparison collected by the CVA and pressure probe at U = 22.6 m/s.

For each successive velocity cases, it can be seen that the hot-wire data is not different from the pressure probe by only a linear shift. This maybe due to the fact that the temperature has drifted a significant amount from the ambient temperature that was observed during the hot-wire calibration. The turbulent boundary layer profile is closely matched by the CVA when compared to the pressure probe data at locations near the edge of the boundary layer. With the suggested 1.5% error per degree Celsius and a

estimated temperature increase of about a degree Celsius in the $U_\infty = 30.2\, m/s$ case, two degrees Celsius in the $U_\infty = 38.3\, m/s$ case, and three degrees Celsius in the $U_\infty = 45.9\, m/s$ case, temperature drift is the main contribution to the error seen in this region. However, the discrepancy that is seen between the CVA and pressure probe data near the surface is most likely due to both the temperature drift and the uncertainty in the starting position. In the region near the flat plate, $\frac{du}{dy}$ is relatively large so a slight uncertainty in the y-location yields a large difference in the velocity.

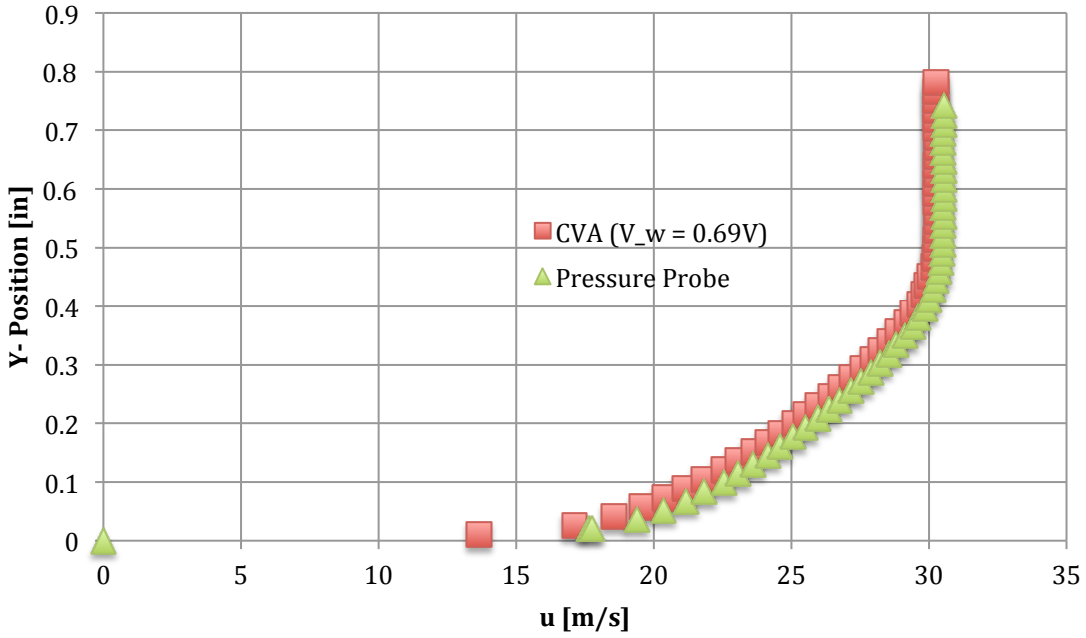


Figure 2.2.5. Turbulent boundary layer profile comparison collected by the CVA and pressure probe at U = 30.2 m/s.

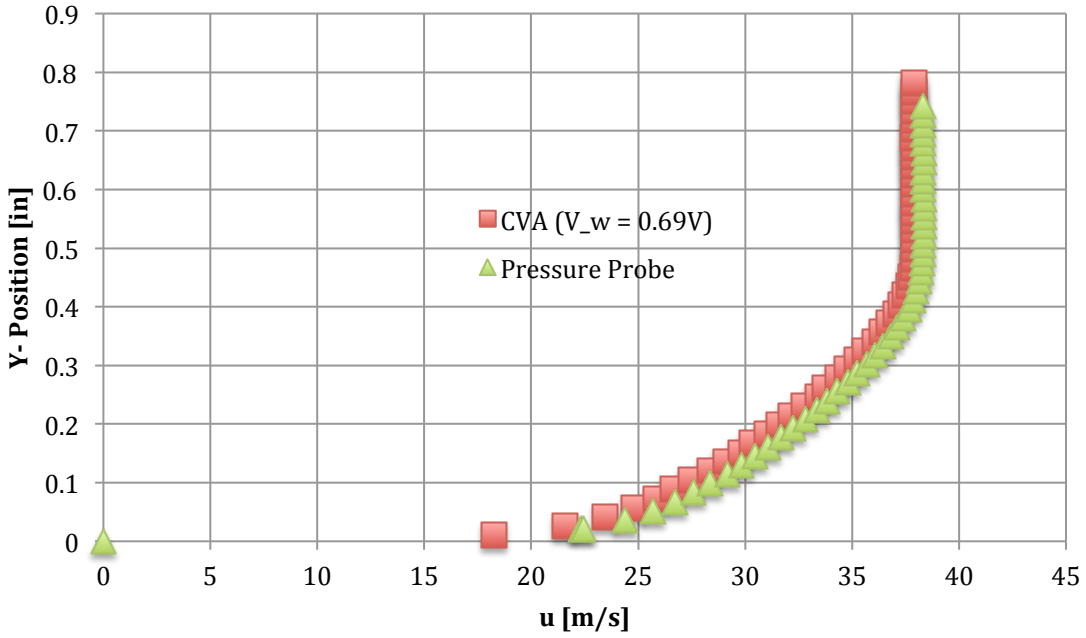


Figure 2.2.6. Turbulent boundary layer profile comparison collected by the CVA and pressure probe at U = 38.3 m/s.

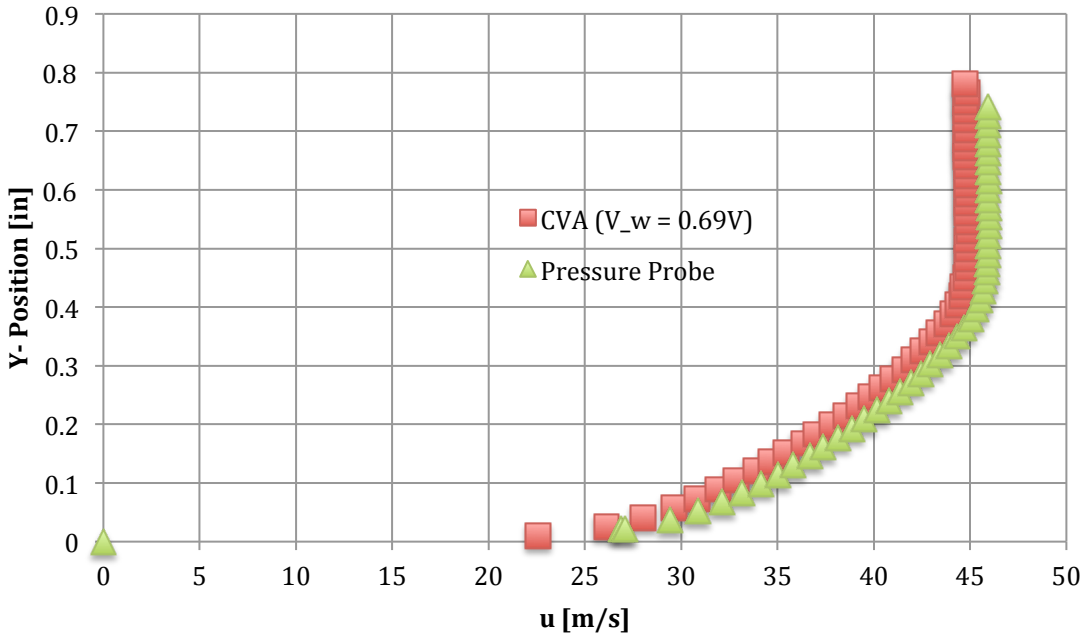


Figure 2.2.7. Turbulent boundary layer profile comparison collected by the CVA and pressure probe at U = 45.9 m/s.

2.3. Variability of CVA Data with Lower OHR

A complementary study was done in addition to the one stated above to understand the effect of OHR when measuring turbulent boundary layer data. With $V_w = 0.69V$, an OHR of 1.65 can be seen at the highest observed velocity and 1.98 at the lowest observed velocity. For the lower OHR study, a wire voltage of $V_w = 0.45V$ was chosen. This wire voltage chosen yields an OHR of 1.33 at the highest observed velocity and 1.53 at the lowest observed velocity.

It can be concluded that lowering the OHR does not yield different results than the test done with higher OHR, see Figure 2.3.1. The lower V_w setting was still able to capture the same turbulent boundary layer profile as the higher V_w setting and had the same trend when comparing the CVA and pressure probe boundary layer velocity profiles. This is expected since OHR only affects the sensitivity and frequency response of the hot-wire probe. Statistically when velocity values are averaged, no matter the magnitude of the fluctuations that occur due to either a performance change in frequency response or the change in sensitivity, the average velocity should stay nearly the same. However, the situation is very different when considering the influence of V_w on the measurement of velocity fluctuations. The OHR does play a very significant role in the velocity fluctuation measurements the hot-wire as will be explained in more detail in Section 2.4.

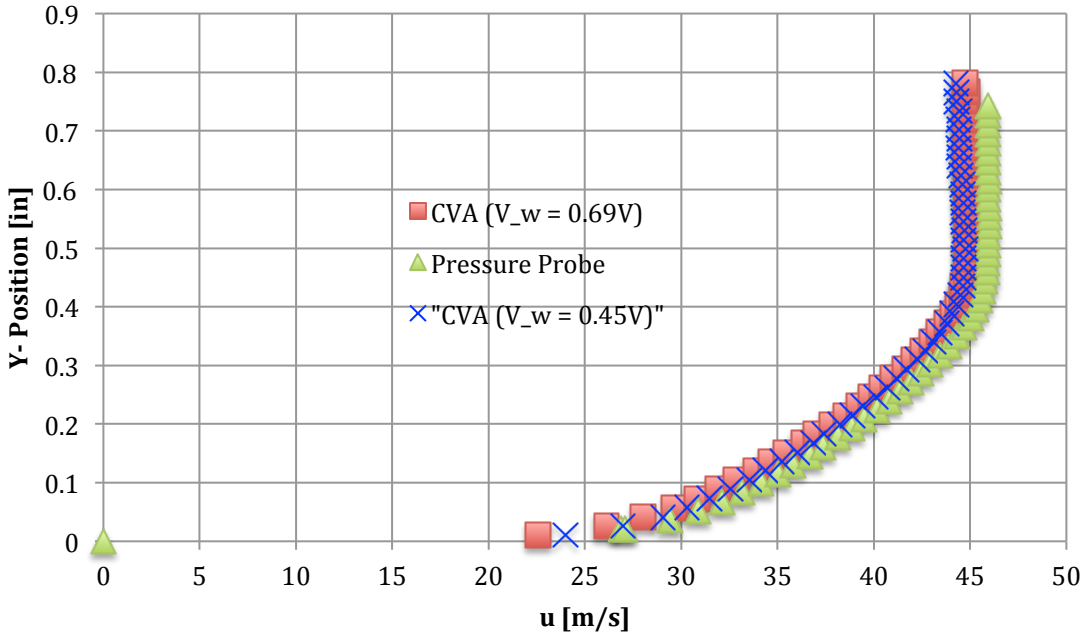


Figure 2.3.1. Turbulent boundary layer profile comparison of a higher OHR and lower OHR at U = 45.9 m/s.

2.4. Turbulence Intensity Measurements

The next step in validating the accuracy of the data taken by the CVA is to compare the velocity fluctuations measured by the CVA with published data. In Figure 2.4.1 – Figure 2.4.4, the boundary layer's velocity fluctuation data from the CVA with $V_w = 0.69V$ for various freestream velocities and the published data from Klebanoff [15] are plotted on top of each other. It is important to note that the Klebanoff's data is collected at a $Re_x \approx 10^7$ where as the CVA data has been measured for Reynolds numbers in the range 0.9 x 10^6 to 1.84 x 10^6, as is shown in Table 2.4.1. Also in the test setup used to run this experiment, as described previously, there is slight favorable pressure gradient so turbulence stresses would be expected to be lower.

Table 2.4.1. Reynolds number for the different test velocities.

U_∞ [m/s]	22.4	30.3	37.9	44.8
Re	9.21 x 10^5	1.25 x 10^6	1.56 x 10^6	1.84 x 10^6

In Figure 2.4.1– Figure 2.4.4, it can be seen that the CVA data follows Klebanoff's data trend fairly well regardless of the freestream velocity. The only inconsistency seen between the data collected through the experiment and Klebanoff's data is that the CVA always under predicts the velocity fluctuations in the $\frac{y}{\delta} \leq 0.55$ region and over predicts the velocity fluctuations in the $\frac{y}{\delta} \geq 0.55$ region. The reason for the velocity fluctuations to be under predicted in the region where $\frac{y}{\delta} \leq 0.55$ is due to the CVA system being limited by a frequency response of 2,200 Hz and the turbulent flow velocity fluctuations can fluctuate up to 8,000 Hz. This is really noticeable in Figure 2.4.1 – Figure 2.4.4. At lower velocities the velocity fluctuations in a turbulent boundary layer contain more energy at lower frequencies and the CVA has a higher frequency response at lower velocities due to a higher OHR. As a result, it allows us to capture a greater spectrum of the fluctuating velocities. However at higher velocities the opposite is true. The velocity fluctuations in a turbulent boundary layer fluctuate at a higher frequency and the CVA has a lower frequency response at higher velocities due to a lower OHR. Therefore the peak at near the surface in Klebanoff's data is, as expected, increasingly undermeasured in the CVA data collected as the freestream velocity is increased.

The over-prediction seen in the CVA data collected in the $\frac{y}{\delta} \geq 0.55$ region is in partial due to the electrical noise present in the operation of the CVA and, to a lesser extent, the resolution of the NI USB-6009. In the freestream region, a $\frac{u'}{U_\infty}$ value of 0 should be observed since it is assumed that the velocity in the freestream is steady. However this is not the case, a noticeable amount of fluctuation in the flow is seen in the

freestream at about 0.6% of freestream velocity. This correlates to the flow varying up to $\pm 0.90\ m/s$ in the freestream for the $U_\infty = 44.8\ m/s$ case.

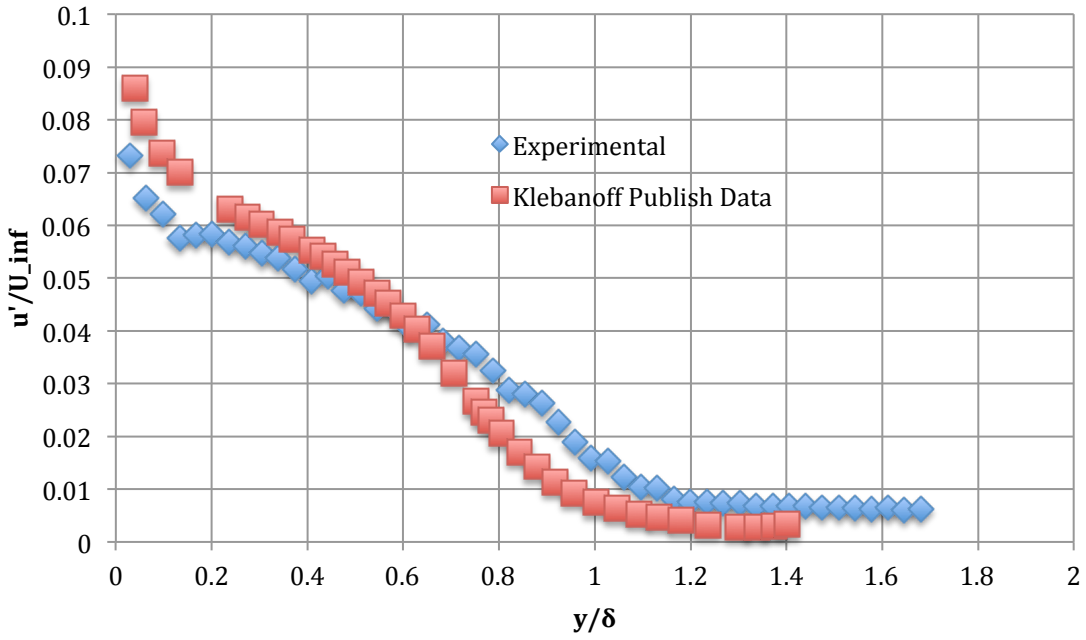


Figure 2.4.1. Comparison of velocity fluctuation data between the CVA and Klebanoff's published data at U = 22.4 m/s.

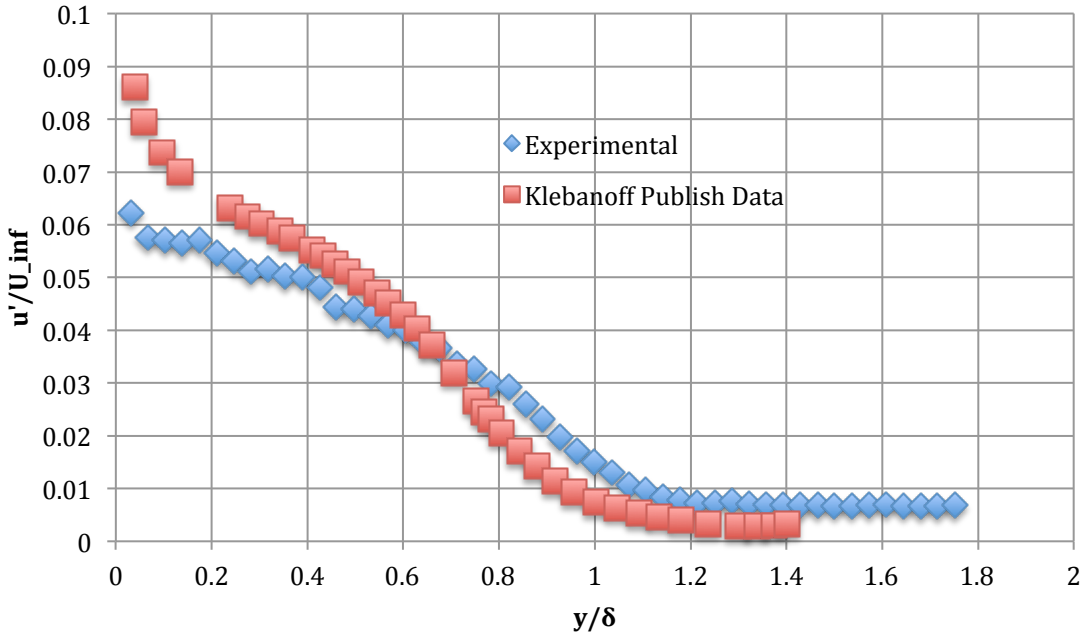


Figure 2.4.2. Comparison of velocity fluctuation data between the CVA and Klebanoff's published data at U = 30.3 m/s.

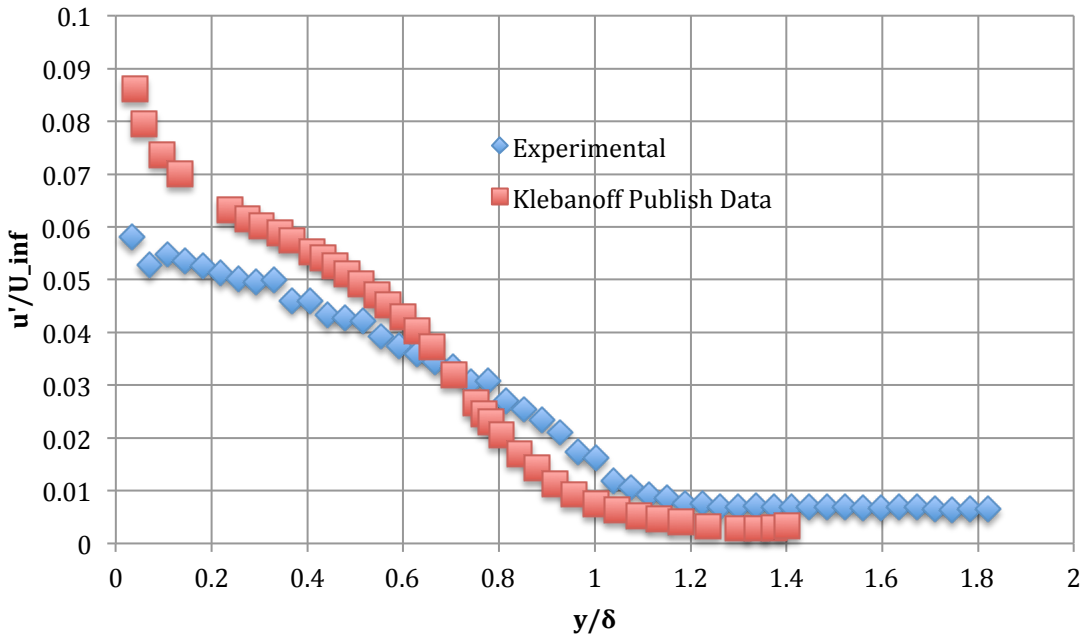


Figure 2.4.3. Comparison of velocity fluctuation data between the CVA and Klebanoff's publish data at U = 37.9 m/s.

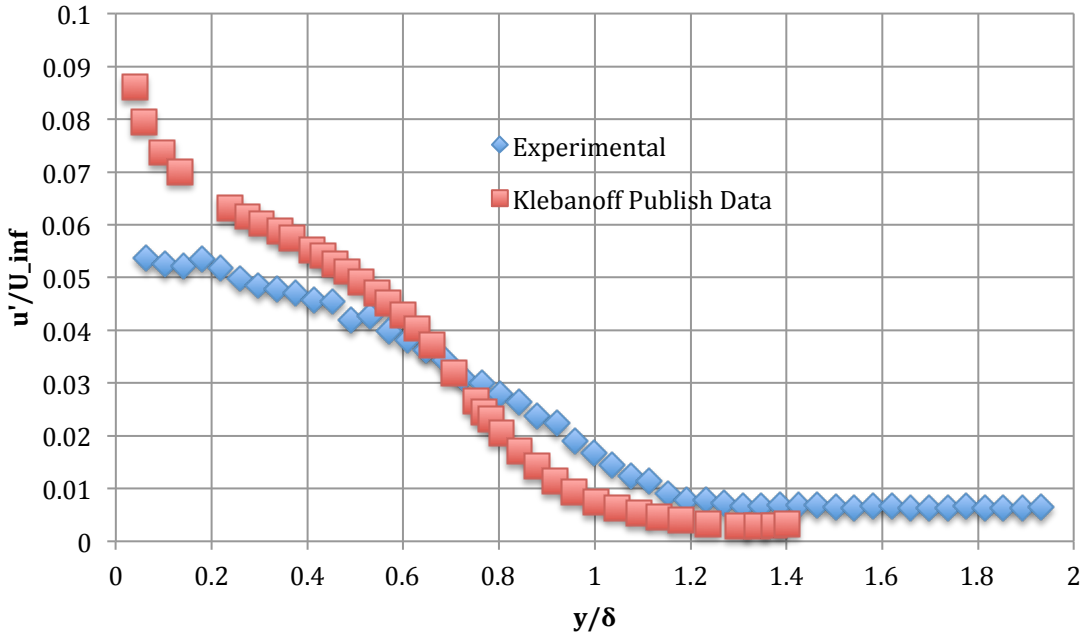


Figure 2.4.4. Comparison of velocity fluctuation data between the CVA and Klebanoff's publish data at U = 44.8 m/s.

2.5. Coles and Hirst Inner Variable Validation

It is also desired to compare the mean velocity profile from CVA to the Coles and Hirst inner variable correlation [16]. The result from plotting the velocity profiles in terms of the inner variable $u^+ = \frac{u}{u_\tau}$ and $y^+ = \frac{yu_\tau}{\nu}$ can be seen in Figure 2.5.1– Figure 2.5.4. From the inner variable correlations, a linear relationship should be seen in the log-law region where, approximately, $35 \leq y^+ \leq 350$. This is seen in the hot-wire data collected from the experiment. The experimental data closely, if not, exactly matches the inner law variable correlations in the log-law region presented by Coles and Hirst. The only discrepancy comes when y^+approaches 350. A reason for the discrepancy between the CVA data and the inner law is due to the approximation use for the starting height of the hot-wire when taking the boundary layer profile. Changing the starting height shifts

the hot-wire data up if a higher starting position is used and shifts the hot-wire data down if a lower starting position is used.

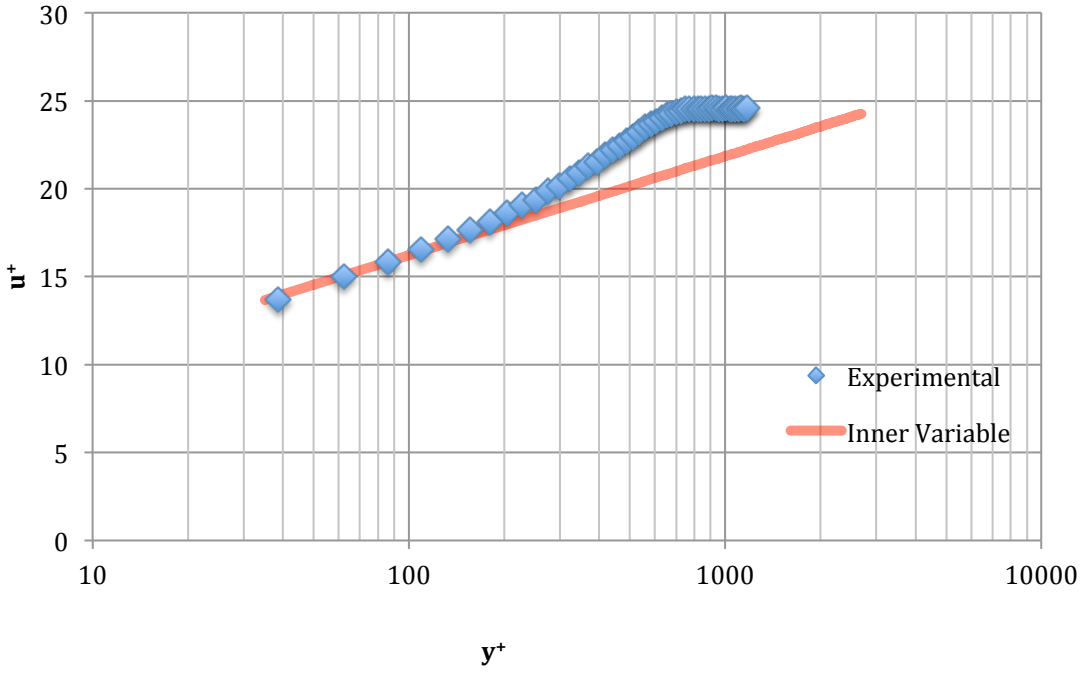


Figure 2.5.1. Velocity profile replotted with inner variables for U = 22.4 m/s.

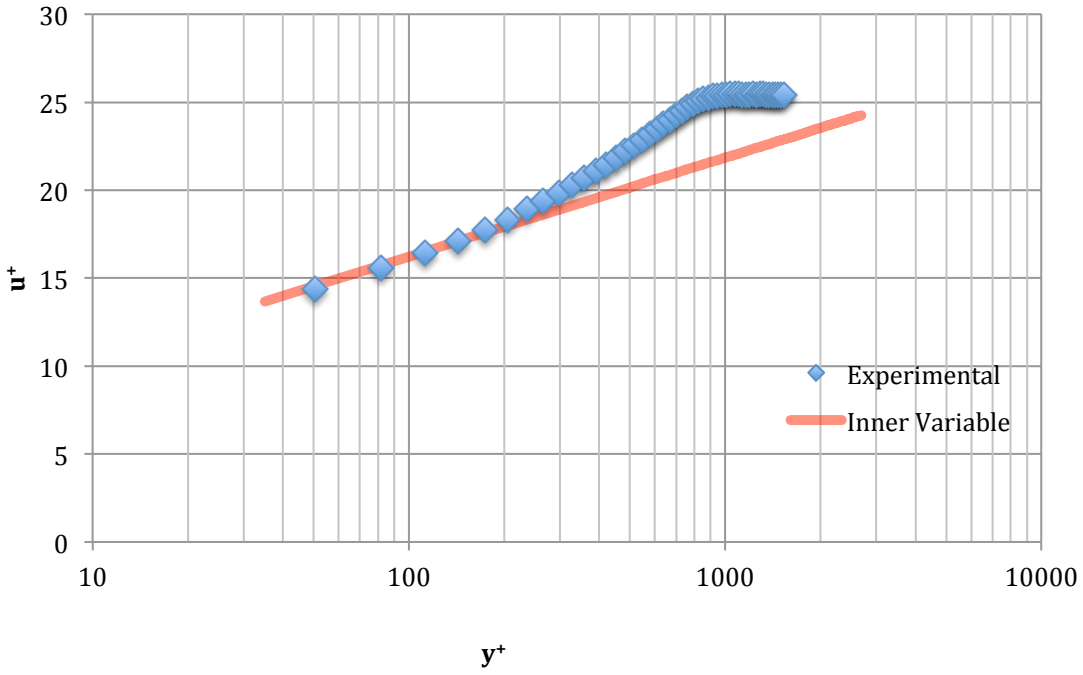


Figure 2.5.2. Velocity profile replotted with inner variables for U = 30.3 m/s.

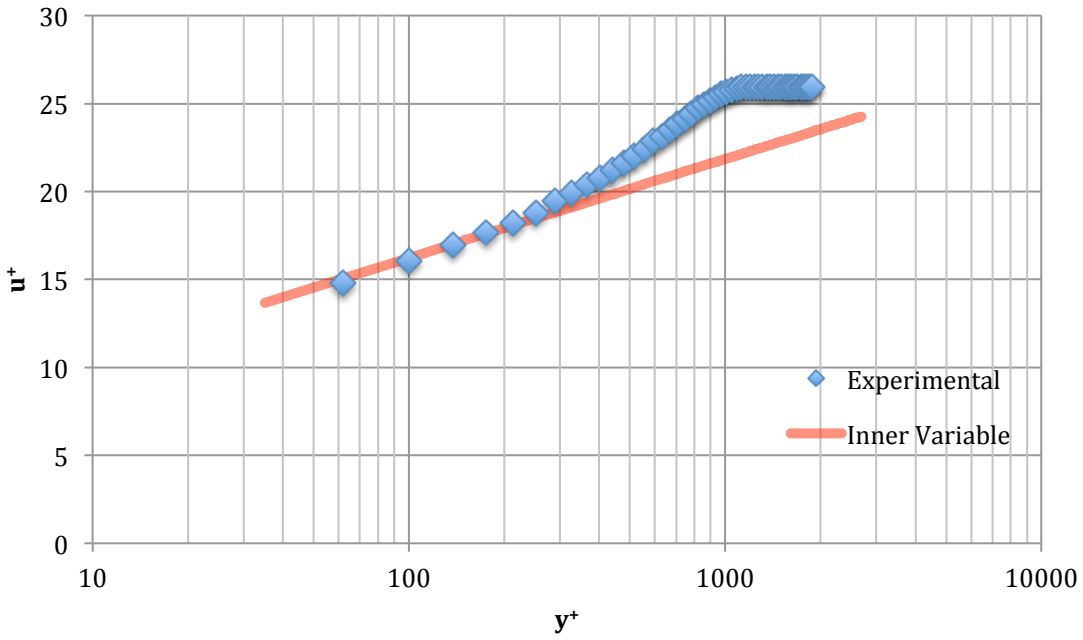


Figure 2.5.3. Velocity profile replotted with inner variables for U = 37.9 m/s.

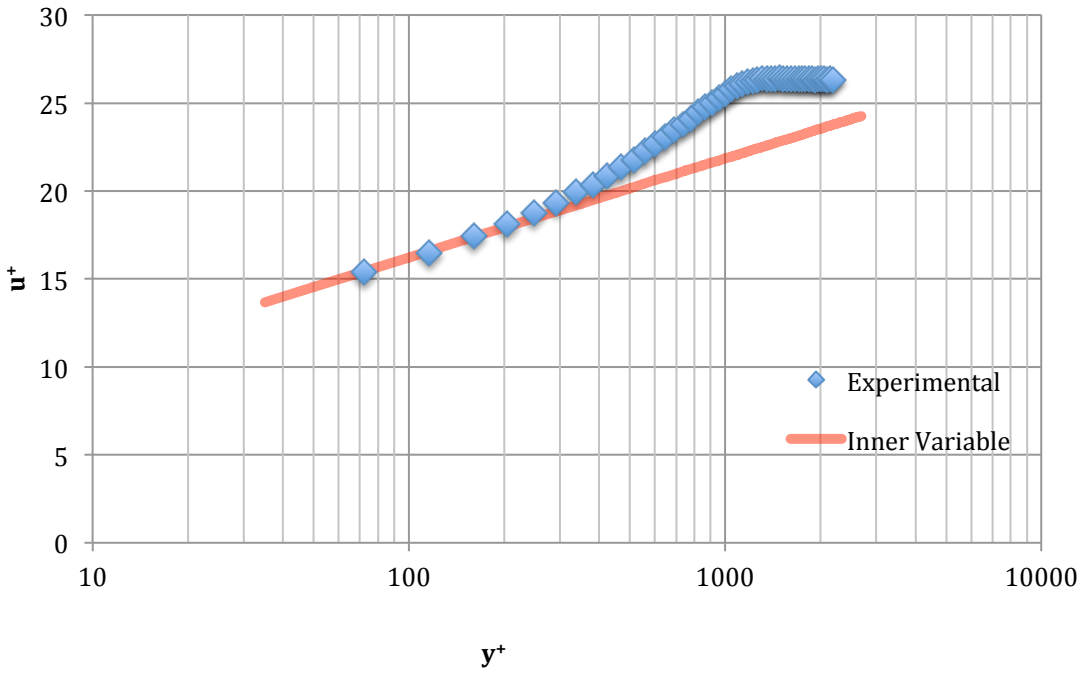


Figure 2.5.4. Velocity profile replotted with inner variables for U = 44.8 m/s.

2.6. Conclusions Concerning CVA Boundary Layer Measurements

From the data shown in the previous sections of this chapter, it is clear that the CVA is able to capture mean boundary layer data as accurately as a proven method as the total pressure probe. The greatest percent error seen between the two methods is 1.2% for the boundary layer thickness, 4.9% for the displacement thickness, 5.9% for the momentum thickness, and 6.4% for the C_f values. The smallest percent error seen between the two methods is 0.4% for the boundary layer thickness, 0% for the displacement thicknesses, 4.0% for the momentum thickness, and 4.6% for the C_f values. Some of these inaccuracies are due to the uncertainty in the starting position of the hot-wire and the flow temperature drift in the wind tunnel while the hour long experiments are performed. However, temperature drift is not prevalent during steady level flight so this should not be a major concern for the operation of the BLDS – CVA in flight test applications. As for the CVA's ability to capture turbulence intensities in the boundary layer, in the slowest velocity case, it can be seen that the CVA does have the frequency response required to capture a wide spectrum of turbulence intensities. This is a result of an increase in frequency response due to a higher OHR at low velocities. This can be changed for higher velocities to achieve the same results seen at lower velocities by increasing the wire voltage, although this would have to be done carefully to avoid the possibility of wire burnout.

With the repeatable results seen with the hot-wire data, it can be concluded that the CVA can be used confidently in measuring boundary layer mean velocity profiles and the corresponding calculated boundary layer characteristics. The next step in developing the CVA for the BLDS will be to make sure the BLDS hardware and software can

support the addition of the hot-wire. The changes made to the BLDS hardware and software to ensure a smooth operation of the BLDS – CVA can be seen in the following chapter.

3. BLDS – CVA HARDWARE AND SOFTWARE

3.1. BLDS – CVA Hardware Components

A new version of BLDS with hot-wire capability has been fully redesigned to meet all the new requirements imposed onto the BLDS with the addition of the CVA. The exterior hardware changes to the BLDS can be seen in Figure 3.1.1 and the interior hardware changes to the BLDS can be seen in Figure 3.3.3. However, even with the new addition, the previous versions of BLDS components are still present. This allows the user to use other probes for measurement of the boundary layer without having to replace what would be the "new" hardware with the "old" hardware. The user would only have to replace the hot-wire probe with the probe of their choice.

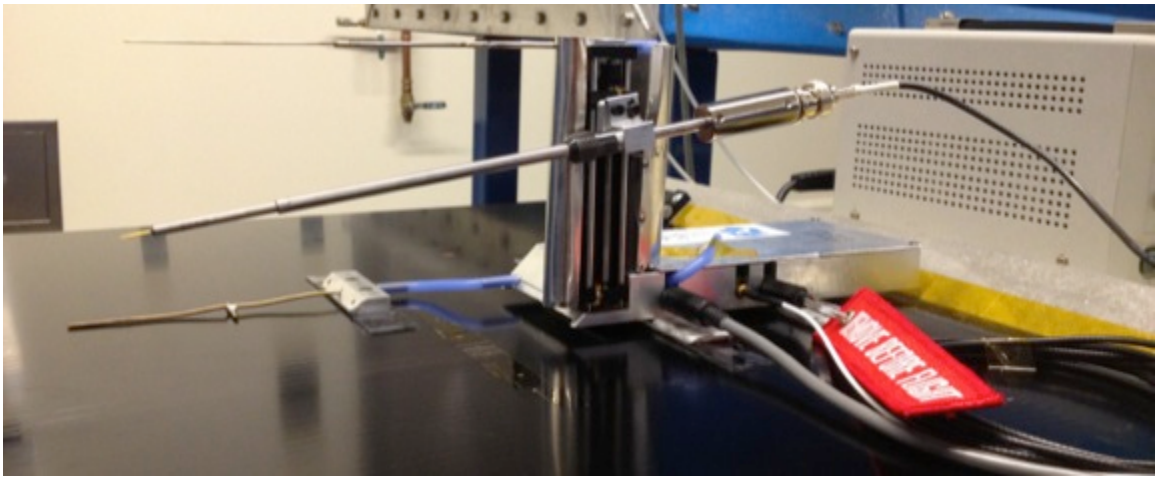


Figure 3.1.1. Redesigned BLDS with CVA capability.

With this interchangeable quality, the new BLDS – CVA houses a similar circuit board as the previous version of the BLDS with the only addition to the circuit board being the removable CVA Daughter Board that can be plugged into a new socket that replaces the stepper motor port of the previous version of the BLDS circuit board. See Section 3.2 for more details of the CVA Daughter Board and Appendix B for instructions to safely operate the CVA.

The motorized stage is identical to the ones used in previous versions as well and operates in the same manner. This specific motor on the BLDS – CVA is a modified version of the National Aperture MM-3M-F and has a 16:1 gear head that allows for precise movement (0.00007813 inches/count) of the probe throughout the boundary layer. It also has limit switches on the top and bottom limits to avoid mechanical failure. The data sheet for all the motor's specification can be seen in Appendix C.

The BLDS – CVA employs a modified TFX–11v2 dual microcontroller subsystem, this will be explained in Section 3.3, and is programmed using TFBASIC programming language. The software's capabilities for this device are explained in Section 3.4. A problem that arises with using the TFX–11v2 data logger hardware with the hot-wire is the limited 2MB memory that comes with the product. In order to measure mean and fluctuating velocity and fluctuating velocity values within the boundary layer data with a hot-wire, hundreds of samples need to be taken at each location. In addition, in order to get a complete boundary layer profile, samples have to be taken at numerous locations within the boundary layer. This amounts to over hundreds of thousands of data samples. With only 2MB of memory, all the 'raw' data cannot be saved; only selected statistical quantities such as average and standard deviation of the data can be saved. This then becomes a problem when converting the voltage value from the hot-wire to velocity values. In previous versions of BLDS with pressure sensors determining the velocity of the flow, the average voltage output from the pressure sensors was saved and processed after download. Only average pressures were needed to compute 1-, 2-, or 3-component mean velocity data from the different BLDS pressure-based anemometers. This makes it possible to use the average voltage values to convert to pressure and therefore find the

corresponding velocity values. However, since the hot-wire relates voltage to velocity through a non-linear calibration function, simply taking the average voltage values and plugging these into a calibration function to get the average velocity values does not work. Theoretically, each data point has to be converted into velocity and then averaged. However, this is not a valid solution as it can take a long time to compute with the microprocessor onboard and the conditions can change in flight during that time. As a result, an alternative solution to this problem needs to be found. This solution can be found in Section 3.5.

One of the major changes done to the hardware when compared to the previous versions of BLDS is the three pressure sensors on board. The two differential pressure sensors used to measure the freestream dynamic pressure and the local dynamic pressure use the All Sensor's 10 Inch G-P4V MINI pressure sensors while the absolute pressure sensor used to measure the local static pressure uses the Honeywell's SSCDRNN015PAAA5 pressure sensors. The data sheets for both models of pressure sensors can be seen in Appendix C. These two different models of pressure sensors differ from previous versions of BLDS by the dynamic range they are able to capture and the environmental conditions they are able to endure. The All Sensor's differential pressure sensor can only measure pressure readings from 0 to 10 inches of water and has a temperature compensator that only results in accurate readings when the ambient air temperature is between -25 - $85℃$. The Honeywell's absolute pressure sensor can measure absolute pressure anywhere between 0 to 15 psia and only outputs accurate pressure readings when the ambient air temperature is between -20 - $85℃$.

This poses a problem for using the BLDS – CVA during flight-testing as HALE aircrafts can reach speeds up to Mach 0.5 and the ambient temperature at the cruise altitude of these aircrafts, 60,000 feet, can reach a temperature as low as $-60℃$. However the implementation of accurate pressure sensors at altitude can be easily done. This only requires looking for pressure sensors that meet the speed and temperature requirements required for flight testing or simply using the pressure sensors on previous versions of BLDS that have been proven to work. The reason why these lower range, lab condition pressure sensors are used is because the tests that are required to run in order test the implementation of the hot-wire and study the problems that arises with the addition of the hot-wire to the BLDS, do not require higher dynamic pressure readings than 10 inches of water and the ambient conditions in lab will never exceed the temperature ranges specified by these manufactures. Using lower ranged pressure sensors also allow for more accurate pressure readings in the lab wind tunnel speed conditions, as they have better sensitivity to small changes. Calibration constants and current draws of the pressure sensors on board the BLDS – CVA can be seen in Table 3.1.1.

Table 3.1.1. Parameters of the pressure sensors on board BLDS - CVA used for design and computation of boundary layer data.

Pressure Sensor	Calibration Constant	Current Draw
Freestream Dynamic Pressure	2.4316 in. H_2O/V	3.9 mA
Local Dynamic Pressure	2.4505 in. H_2O/V	3.8 mA
Local Static Pressure	3.7567 psi/V	2.3 mA

Another component that has been changed in this version of BLDS is that the battery used to power the device is a commercial non-rechargeable 9-Volt battery. A 9-Volt battery with Lithium Manganese Dioxide ($LiMnO_2$) chemistry is suggested due to the high current draw of the device and the minimum of 8V required by the BLDS –

CVA. The Energizer 9-Volt $LiMnO_2$ battery was used for all experiments completed in Chapter 4 and the product specification can be seen in Appendix C. It should be noted that if the BLDS – CVA were to be flown at altitude, a battery that can withstand the extreme conditions would be required be used instead. This 9-Volt battery can be easily be replaced by the high performance battery with no additional changes.

Lastly, the hot-wire probe replaces the pressure probe on the BLDS – CVA. The hot-wire probe used for this thesis is the TSI 1210 – T1.5 hot-wire probe. This is the same hot-wire probe described in Section 2.1. This hot-wire probe is mounted on the TSI-1150AA hot-wire probe holder and the hot-wire probe holder is mated onto the stage through a unique part manufactured in-house [10]. The hot-wire probe holder is connected electrically to the CVA Daughterboard through a BNC-MCX cable. Note that any hot-wire probe can be used with this device as long as the BLDS – CVA can handle the power requirements of the hot-wire probe. With the current setup of the BLDS – CVA, the maximum current draw of this device is 180 mA. The CVA Daughterboard draw 53 mA of current, leaving about 120 mA left for the hot-wire probe. If the sensor resistance exceed $R_\infty = 8.3\ \Omega$ the maximum current draw of the CVA Daughterboard will be less than 120 mA.

3.2. CVA Daughterboard

The CVA Daughterboard is the required hardware that allows the hot-wire probe to be used on the BLDS. It is connected to the BLDS electronics through a new socket that replaces the stepper motor port of the previous version of the BLDS circuit board. The hot-wire probe mounted onto the hot-wire probe holder can then be connected to the CVA Daughterboard through a BNC-MCX cable. Although the CVA Daughterboard is

made for use with the BLDS, it can also be used on the benchtop via the CVA Daughterboard Box. Figure 3.2.1 shows the interface of the CVA Daughterboard Box with the CVA Daughterboard mounted and the available options for the user. The CVA Daughterboard is mounted correctly on the CVA Daughterboard Box when the MCX cable input is located on the top right hand corner facing away from the buttons and switches. Note, that unlike the CVA Benchtop Box, the CVA Daughterboard only outputs the $V_{w,out}$, I_w, and $V_{w,in}$ values. Therefore any additional values desired, aside from the three values specified here, need to be calculated using the equations that govern the CVA circuit [11].

Figure 3.2.1. CVA Daughterboard Box interface with CVA Daughterboard mounted.

On the CVA Daughterboard Box there is a display that shows the $V_{w,out}$ and I_w readings from the hot-wire when the *-Vw Out* and *-Iw* buttons are suppressed, respectively. There are also four binding post available to connect banana cables to read the values measured by the hot-wire probe from the CVA Daughterboard using either a

Digital Voltmeter (DVM), Oscilloscope, or other device of the user's choice. All the readings outputted by this device are relative to a reference voltage. In order to convert the voltage outputs to the actual voltage seen across the hot-wire and the current across the hot-wire in milliamps, a constant calibration constant needs to be multiplied to the output values. The actual V_w seen across the hot-wire can be found using Equation (3.2.1),

$$V_{w,out} = \frac{V_{w,measured}}{5\left(\frac{Volts}{Volt}\right)}. \tag{3.2.1}$$

And the current across the hot-wire in milliamps can be found using Equation

$$I_w = \frac{I_{w,measured}}{41.6\left(\frac{Volts}{mA}\right)}. \tag{3.2.2}$$

There are also three switches on the interface of the CVA Daughterboard Box. The *Pwr* switch is used to power the CVA Daughterboard, the *Sensor* switch is used to allow current to flow to the hot-wire probe, and the *Stop* switch is used to help set the V_w value for the hot-wire probe. Note that it is crucial to leave the *Sensor* switch off when a hot-wire probe is not connected to the CVA Daughterboard to avoid high current flow through the electronics and to set the V_w before powering the hot-wire to avoid hot-wire burnout.

To set the V_w value, a good understanding of the CVA Daughterboard Box's operation is required. When the CVA Daughterboard Box is first powered, the default V_w voltage is $V_w = 0.495V$ and the mode of the CVA Daughterboard Box is set to increase the V_w voltage. In order to increase the V_w voltage, simply pressing the red $V_w$button once and this will result in a $\Delta V_w \approx 0.016V$. This can be increased until the maximum value of $V_w = 1.000V$ is reached. However, as stated previously, any hot-wire with a $R_\infty = 8.3\Omega$

or greater will cause an inaccurate reading of I_w as the CVA Daughterboard cannot draw more current than 120 mA. To decrease the V_w voltage, the CVA Daughterboard Box will need to be set to decrease V_w voltage mode. To do this, have the *Stop* switch set to off and then switch the *Stop* switch to *Step* with the red V_w button suppressed. This will result in a $\Delta V_w \approx -0.016V$. This can be reduced until the minimum value of $V_w = 0.000V$ is reached.

The operation of the CVA Daughterboard on the BLDS – CVA is identical to the CVA Daugtherboard Box except all the functions are done through the software instead of manually by the user. To install the CVA Daughterboard onto the BLDS – CVA from the CVA Daughterboard Box, simply dismount the CVA Daughterboard from the CVA Daughterboard Box and mount the CVA Daughterboard to the BLDS – CVA with the MCX cable input fitting through the 2nd slot from the front on the side of the BLDS - CVA.

3.3. Change to A/D Reference on TFX–11v2 Circuit Board

In a preliminary turbulent boundary layer velocity test with the BLDS – CVA with the same test setup as described in Section 2.1 but with the hot-wire probe placed at a location of $x = 23.4\ inches$, it was observed that velocity fluctuation measured by the BLDS – CVA were 3 to 4 times that of what should be seen in the freestream, see Figure 3.3.1. This was also seen while testing the BLDS – CVA ability to capture a laminar boundary layer velocity profile where the velocity fluctuations readings are high throughout both the laminar boundary layer and in the freestream, see Figure 3.3.2. The test setup for the laminar boundary layer velocity test is similar to turbulent boundary layer velocity profile test as described here previously but with the trip removed.

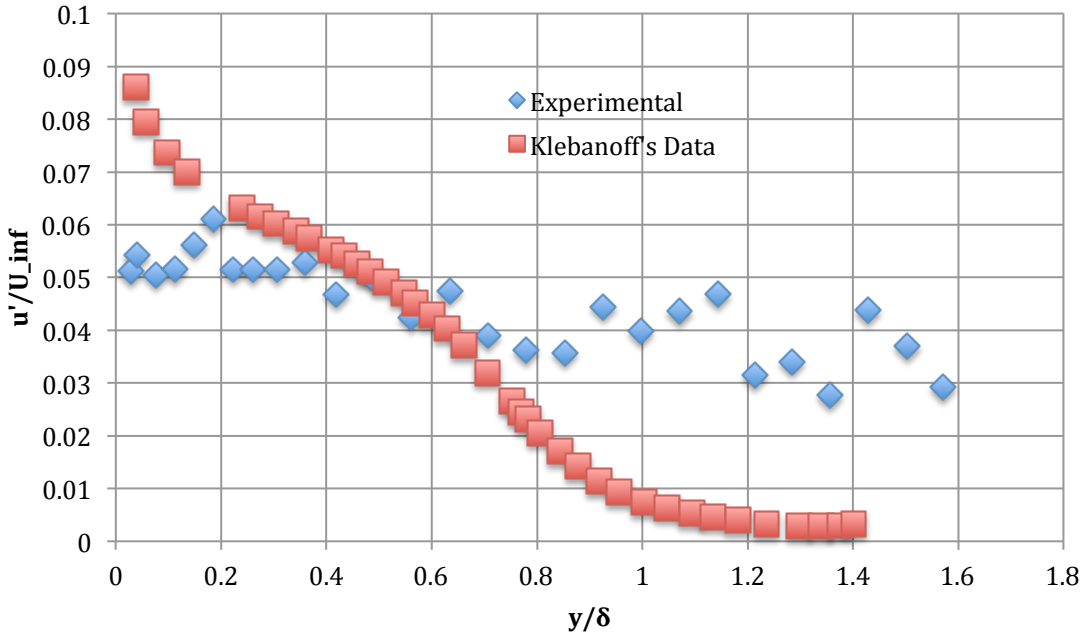


Figure 3.3.1. Velocity fluctuation data from a turbulent boundary layer velocity profile with unclean A/D reference on BLDS – CVA compared to Klebanoff's Data at U = 46 m/s.

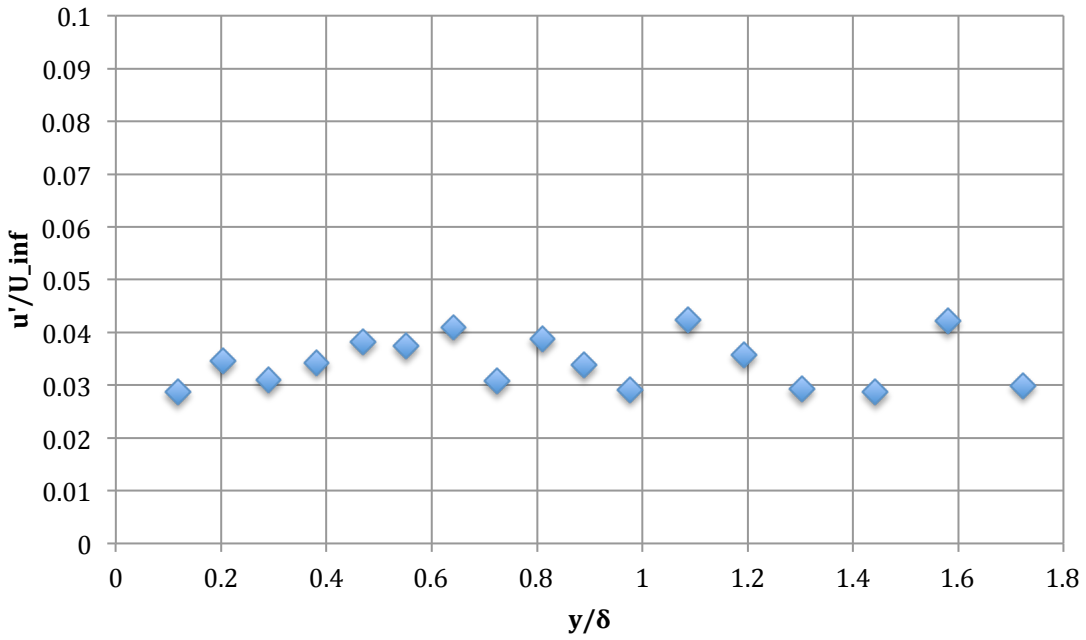


Figure 3.3.2. Velocity fluctuation data from a laminar boundary layer velocity profile with unclean A/D reference on BLDS - CVA at U = 46 m/s.

From the benchtop test, a $\frac{u'}{U_\infty}$ value of less than 0.01 was expected in the freestream, similar to the one published by Klebanoff [15] for the turbulent case. However this was not the case. This called for an in-depth study of the BLDS – CVA to find the source of the problem and to find a solution to the problem.

It was found that the Analog/Digital (A/D) reference on the TFX–11v2 was the source of the problem. The onboard A/D reference used a linear regulator to provide the 5V A/D reference. However, the linear regulator was not able to provide a clean, flat, 5V reference needed to ensure accurate data measured by the hot-wire probe. This phenomenon was found by reading the ADVCC signal output from the ADVCC pin from the TFX–11v2 with a Tektronix TDS 2002 Oscilloscope, see Appendix C for the pin layout of the TFX–11v2. With the oscilloscope it was found that a spike in the 5V A/D reference occurred every 10 milliseconds with values peaking at $+5.015mV$ and $-5.030mV$. This caused the velocity fluctuation or standard deviation of the I_w voltage output to be around $8 - 10mV$.

To solve this problem, the onboard A/D reference that the TFX–11v2 board uses was replaced by the regulated 5V power supply from the BLDS – CVA power source. This was done by removing the R27 0 Ω resistor that connects the output of the on-board linear regulator to the ADVCC pin and connecting the ADVCC pin to the regulated 5V power supply on the BLDS – CVA. See Appendix C for the schematic of the TFX–11v2 board and the location of the R27 resistor.

As can be seen in Figure 3.3.3 below, the connection is currently made with an electrical wire with insulated pin terminals on both ends. However, a more permanent fix will be implemented in the next iteration of the BLDS – CVA.

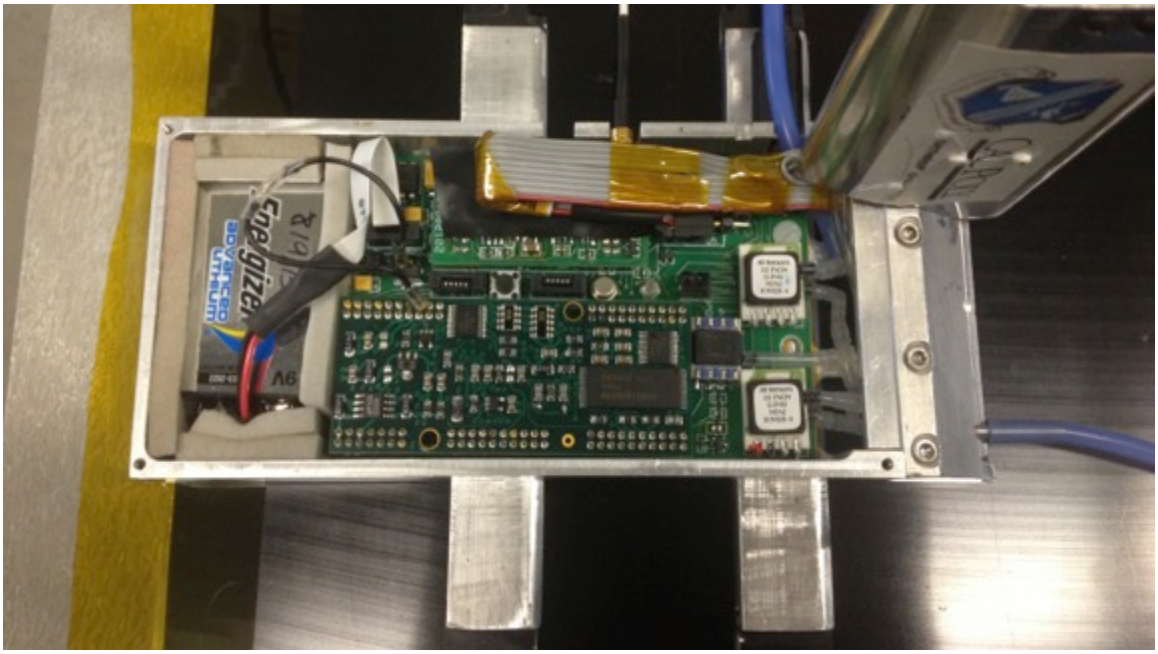


Figure 3.3.3. Temporary solution for the new A/D reference implemented onto the BLDS - CVA.

The test result from the hardware change was very favorable. The standard deviation of the I_w voltage output dropped significantly to $0-5mV$ in the freestream from $8-10mV$. The new I_w voltage output results made the velocity fluctuation values match the expected values more closely, similar to the values seen in the benchtop test in Section 2.4, where the values are only slightly off from published data due to electrical noise.

3.4. Software for BLDS – CVA and Operation

With the addition of the CVA hardware onto the BLDS, there was a need to create new software that would allow the electronics to communicate with the mechanical parts. As always with previous versions of BLDS, two software programs were developed specifically for the BLDS – CVA. The first program that was created is the BLDS – CVA Hardware Test Program. This program allows the user to check that all parts on the BLDS are functioning correctly. This ensures the BLDS – CVA will be working properly

during the flight test and the data retrieved from the device is accurate. This program can test the pressure probe version of the BLDS – CVA as well. The second program allows the user to command the BLDS – CVA to take boundary layer profile measurements. There is also a third program that was created to allow the use of the pressure probe with the BLDS - CVA. This was needed because the pin layouts were changed from the previous version of BLDS. The full programs are shown in Appendix D.

The BLDS – CVA Hardware Test Program is setup in a menu structure format. When it is first uploaded onto the TFX–11v2 microcontroller, the software helps initiate the BLDS – CVA by setting all the pins to "LOW" to ensure that the program starts from the same pin settings each time it is turned on, checks if the data saved from previous uses of the BLDS – CVA is removed to ensure that the offloaded document is correctly written, and initialize the CTRL – C function to allow the user to force quit the program. After the initialization is completed a menu prompt appears in the command window with the option to:

1. Test the sensors on board and the battery
2. Test the hot-wire
3. Test the stage assembly
4. Quit the program

The user then can select any of these sub-menus to test the desired function of the BLDS – CVA. Quitting the program stops the user ability to communicate with the BLDS – CVA and sets all pins to "LOW" to ensure that no power is supplied to the BLDS – CVA when it is not being used. Note that none of the values measured by the BLDS – CVA using the Hardware Test Program will be saved.

Under the "Pressure Sensors/Battery" sub-menu, the user will find the various options to test all the sensors. These options include:

1. Reading a sensor for a user specified duration and display the average readings
2. Read a specific sensor and display the bit readings
3. Read a selected sensor and display the average, min-max, and standard deviation values
4. Flash the LED
5. Exit the "Pressure Sensors/Battery" sub-menu and return to the main menu

The sensor options from this menu include the pressure sensors, battery voltage, and temperature reading from the temperature sensor.

Under the "Hot-Wire" sub-menu, similar functions seen in the "Pressure Sensors/Battery" sub-menu can be seen here. The options available in this sub-menu are:

1. Set V_w and read the V_w the BLDS – CVA is outputting to the hot-wire
2. Read all 3 CVA outputs, $V_{w,in}$,$V_{w,out}$, and I_w, for 5 seconds and displays the average values*
3. Read a selected sensor and display the raw readings*
4. Read a selected sensor and display the average, min-max, and standard deviation*
5. Calculate the suggested V_w based on ambient/testing conditions
6. Exit the "Hot-Wire" sub-menu and return to the main menu.

Note: * Hot-wire needs to be attached when running these options to avoid high amount of current flowing throughout the board.

In the "Hot-wire" sub-menu, any option that prompts the user to set the V_w value will be rounded down to the next closest multiple of 0.016 volts. This prevents setting V_w too high, which could results in hot-wire burnout.

One problem that was encountered while writing the software for the hardware test program for the BLDS – CVA is that option 5 in the "Hot-wire" sub-menu required the computation of $Re^{0.45}$ and $\left(\frac{T_f}{T_\infty}\right)^{0.17}$. TFX–11v2 does not have the ability to do mathematical computations where a number needs to be raised to a power. So a mathematical approximation was needed to compute these numbers [17]. The mathematical approximation used for $Re^{0.45}$ is shown in Equation (3.4.1)

$$Re^{0.45} \approx 1 + \sum_{n=1}^{\infty} \frac{(0.45 \ln Re)^n}{n!}. \tag{3.4.1}$$

A similar approximation is used for $\left(\frac{T_f}{T_\infty}\right)^{0.17}$ as well and this can be seen in Equation (3.4.2)

$$\left(\frac{T_f}{T_\infty}\right)^{0.17} \approx 1 + \sum_{n=1}^{\infty} \frac{\left(0.17 \ln\left(\frac{T_f}{T_\infty}\right)\right)^n}{n!}. \tag{3.4.2}$$

In order for these values to be accurate to five percent of the exact values across the Reynolds number and temperature ratio expected, the Reynolds number term needs 6 terms in the series and the temperature ratio term needs 3 terms in the series. Using 6 terms for the Reynolds number term yields a maximum percent error of 4.9% and using 3 terms for the temperature ratio term yields a maximum percent error of 2.9%. The goodness of fit for the Reynolds number term and temperature ratio term can be visually seen in Figure 3.4.1 and Figure 3.4.2, respectively.

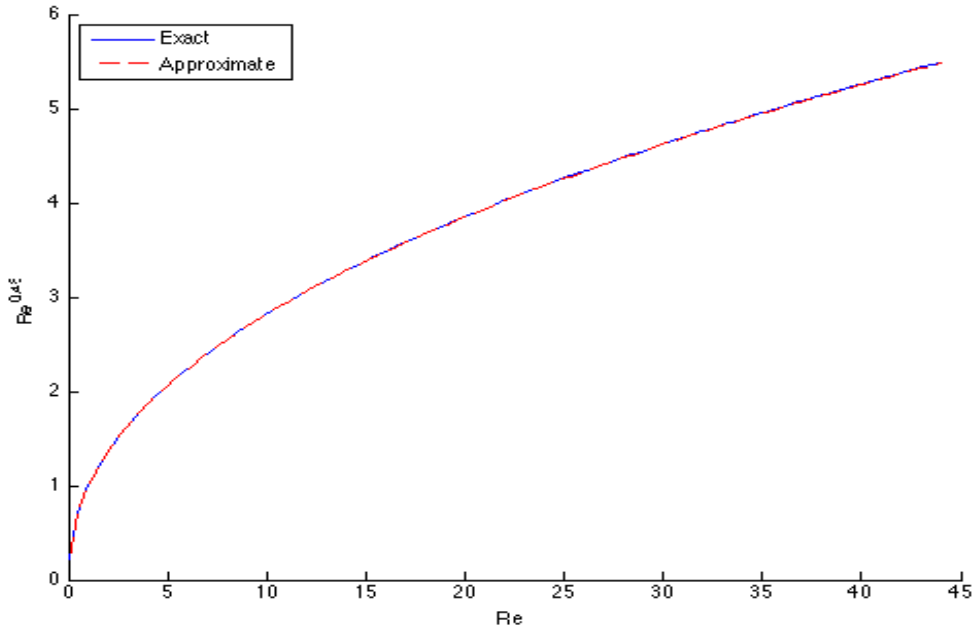


Figure 3.4.1. Goodness of fit of the Reynolds number term between the exact and approximation used in the BLDS - CVA programs.

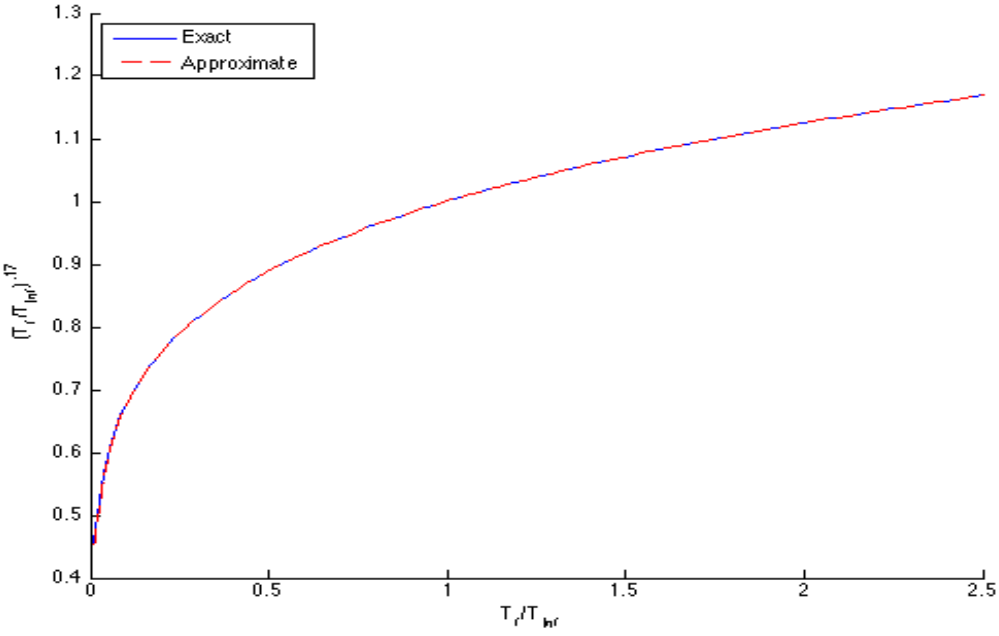


Figure 3.4.2. Goodness of fit of the temperature ratio term between the exact and approximation used in the BLDS - CVA programs.

Note that the equations used to compute the suggested V_w are from Collis and Williams [14]; their model assumed the flow field is two-dimensional, no heat transfer by

radiation, no heat transfer through the prongs that hold the hot-wire, and no change in heat transfer physics when the hot-wire is near a surface. Therefore the V_w values computed by the program will be less than the nominal V_w value that would allow for the nominal OHR and the best frequency response and sensitivity that the hot-wire can achieve given the flow conditions entered by the user.

Lastly the "Stage" sub-menu allows for various options to move the stage up and down through Pulse Width Modulation (PWM) for increased accuracy of stage movements. These options include:

1. Move stage in selected direction for specified number of counts
2. Move stage to the upper or lower limit
3. Move stage in selected direction for a specified time
4. Move probe traverse up or down on each user click for a specified number of counts (Note: limit switches are not activated)
5. Exit the "Stage" sub-menu and return to the main menu

These options can be used as needed to install the hot-wire probe and to make sure that when the hot-wire probe is fully lowered to the lower limit of the stage, the hot-wire does not touch the surface. The stage should be set to the highest limit before switching to the second program so the hot-wire calibration can be completed.

The second of the two programs written is the program that is used to command the BLDS – CVA to take measurements for a boundary layer velocity profile. It is important to note that this version of the program does not have the autonomous flight testing capabilities. Similar to the hardware test program, it has a menu structure. When this program is uploaded to the TFX–11v2, it performs the same initialization as

explained early with the hardware test program. Then it checks to make sure that the battery has the required voltage for the BLDS – CVA to function properly throughout the experiment. It prints out the battery voltage to the user in the command window and if the value is less than 8V, the battery should be replaced. Afterwards it creates the document that will be offloaded along with the boundary layer profile data gathered. It asks for the date and time from the user to use as a reference. After this is all completed, it jumps into the main menu.

In the main menu the option to go to the "Task Menu", "Add General Notes", and "Exit Program" is available. The general notes are saved to the data file and the exit program option performs the exact same function as the hardware test program as stated before. Under the "Task Menu" option the user can find the ability to:

1. Perform a battery check (stored in data file)
2. Perform a freestream check for 10 seconds (not stored in data file)
3. Take pressure windoff data for 10 seconds (stored in data file)
4. Take hot-wire profile (stored in data file)
5. Perform a hot-wire calibration (stored in data file)
6. Add notes specific to the test (stored in data file)
7. Exit task menu and return to main menu

Similar to the hardware test program, any option that prompts the user for the V_w voltage will be rounded down to the next closest multiple of 0.016V to prevent hot-wire burnout.

In all cases the pressure windoff data needs to be taken, this allows the user to calculate the freestream speed of the flow for the calibration of the hot-wire. After that is completed, the hot-wire calibration should be performed by selecting option 5. Option 5

will prompt for the value of V_w to use for the calibration. This value should be known beforehand to ensure hot-wire burnout does not occur; this value will be saved in the data file. In addition, it will ask for notes regarding this calibration measurement and these will be saved in the data file as well. Then a measurement is taken and the Pitot tube, static pressure, temperature, battery voltage, $V_{w,out}$, I_w, $V_{w,in}$, and the number of samples are saved into the data file. Each calibration measurement takes 10 seconds. This can be repeated until the number of desired calibration points is taken for the calibration curve.

Once the calibration is completed a boundary layer profile can be taken with the hot-wire. This option will prompt for the test parameters, which include the number of seconds to wait before each profile, number of seconds to read data, number of counts to move near wall, multiplier used once off the wall, max step increment, number of profiles, maximum encoder counts, and will suggest a V_w value using the same process specified in the discussion of the hardware test program and prompt for a V_w setting. All of these parameters are saved in the data file.

The number of seconds to wait before each profile is the number of seconds it will wait initially and after each profile is complete before beginning the next boundary layer profile measurements. The number of seconds to read data is the number of seconds the hot-wire will measure at each location in the boundary layer profile. Note: Ten seconds is the minimum duration needed to collect enough samples for an accurate average voltage and standard deviation of the voltage. The number of counts to move near wall is the number of counts the hot-wire will move for the first eight points; for subsequent points, the number of steps to move will be multiplied by the user-specified multiplier value until it reaches the maximum step increment. This will continue until the total distance moved

reaches the maximum encoder count. For this version of BLDS – CVA with the 16:1 stage, there are 12,800 encoder counts per inch and a maximum range of 2 inches.

After the test parameters are inputted into the BLDS – CVA, the BLDS – CVA will lower the stage to the lower limit and begin the process in taking a boundary layer profile. It first takes a measurement at the lowest point and then moves on to the next position. To reduce the current draw from the battery, the CVA portion of the BLDS – CVA is turned off when the hot-wire is not measuring data. At each point, the data saved to the data file is the date and time that the data measurement was taken, the voltage output from the Pitot pressure transducer, the voltage output from the static pressure transducer, the temperature from the temperature sensor, the battery voltage, the encoder count, the $V_{w,out}$, the I_w, the $V_{w,in}$, the I_w voltage standard deviation, and the number of samples taken at that position. When the boundary layer profile taken is completed, it returns to the stage's lower limit.

The program for BLDS – CVA with the pressure probe capability is exactly the same as the BLDS – CVA hot-wire program except that the pressure probe program does not have option 5 in the "Task Menu" sub-menu, as it is not needed, and that option 4 in the "Task Menu" is rewritten so the microcontroller knows to measure the boundary layer data from the total pressure probe and not the hot-wire.

3.5. Mathematical Formulation to Deal with Limited Memory Space

To deal with only being able to compute boundary layer data with averaged voltage values and standard deviation of the voltage values, a mathematical formulation was developed to accurately convert these voltage values into velocity values.

Starting with the general hot-wire calibration function, see Chapter 4 for more detail.

$$u = P + QE^k, \tag{3.5.1}$$

where u is the instantaneous velocity and E is the instantaneous voltage output. If Reynolds' Decomposition was performed where

$$u = \bar{u} + u' \tag{3.5.2}$$

and

$$E = \bar{E} + E' \tag{3.5.3}$$

and the equation is time averaged, then

$$\overline{\bar{u} + u'} = \bar{P} + \overline{Q(\bar{E} + E')^k}. \tag{3.5.4}$$

Using the binomial expansion

$$(\bar{E} + E')^k = \bar{E}^k + k\bar{E}^{k-1}E' + \frac{k(k-1)}{2!}\bar{E}^{k-2}E'^2 + \frac{k(k-1)(k-2)}{3!}\bar{E}^{k-3}E'^3 \ldots, \tag{3.5.5}$$

where the 1st order, 3rd order, and higher odd order fluctuating voltage terms (E', E'^3, and etc.) equal zero when time averaged. This leaves only the 2nd, 4th, and higher even order terms. The 4th order and higher even order terms can be neglected due to the expected E' values being much, much smaller than $\bar{E}$ terms in the experiments that will be ran using the BLDS – CVA. This leaves Equation (3.5.6) when time averaged,

$$(\bar{E} + E')^k \approx \bar{E}^k + \frac{k(k-1)}{2!}\bar{E}^{k-2}\overline{E'^2}. \tag{3.5.6}$$

Plugging Equation (3.5.6) into Equation (3.5.4) yields,

$$\bar{u} = P + Q\left(\bar{E}^k + \frac{k(k-1)}{2!}\bar{E}^{k-2}\overline{E'^2}\right)^k, \tag{3.5.7}$$

where $\overline{E'^2}$ is the squared of the voltage standard deviation. The explicit Equation (3.5.7) can then be used to compute the averaged velocity given the average voltage, the

standard deviation of the voltage reading from the BLDS – CVA, and the calibration constant values from the hot-wire calibration. The method used to find the calibration constant can be found in Chapter 4.

To get the fluctuating velocity term given the averaged voltage and standard deviation of the voltage, manipulation of Equation (3.5.4) is needed. Squaring both sides of Equation (3.5.4) yields,

$$\overline{(\bar{u} + u')^2} = \overline{(P + Q(\bar{E} + E')^k)^2}. \tag{3.5.8}$$

Expanding Equation (3.5.8) results in,

$$\bar{u}^2 + \overline{u'^2} = \overline{P^2 + 2PQ(\bar{E} + E')^k + Q(\bar{E} + E')^{2k}}. \tag{3.5.9}$$

Applying the same binomial theorem expansion from Equation (3.5.5) to both $(\bar{E} + E')^k$ and $(\bar{E} + E')^{2k}$ results in,

$$\sqrt{\overline{u'^2}} = Qk\bar{E}^{k-1}\sqrt{\overline{E'^2}}. \tag{3.5.10}$$

The explicit Equation (3.5.10) can then be used to compute the fluctuating velocity given the mean voltage and standard deviation of the voltage from the BLDS – CVA and the calibration constants from the calibration of the hot-wire. The method used to find the calibration constant can be found in Chapter 4.

To assess the accuracy of the approximate method described above, this method was applied to a set of experimental turbulent boundary layer velocity profile data and compared to values that were computed by converting each voltage sample into velocity and then averaged and then taking the standard deviation or in other words the direct method. The tabulated results for all the speeds can be seen in Table 3.5.1 and the turbulent boundary layer velocity profile and the fluctuating velocity results for the $U_\infty = 45\ m/s$ case can be seen in Figure 3.5.1 and Figure 3.5.2 respectively.

Table 3.5.1. Turbulent boundary layer data comparison from direct method versus the mathematical approximation method.

Speed [m/s]	Method	δ (99%) [in]	δ^* [in]	θ [in]	C_f [in]
22	Direct	0.458	0.085	0.058	0.00332
	Approximation	0.458	0.085	0.058	0.00332
30	Direct	0.439	0.082	0.056	0.00309
	Approximation	0.439	0.082	0.056	0.00309
38	Direct	0.422	0.079	0.054	0.00297
	Approximation	0.422	0.079	0.054	0.00297
45	Direct	0.405	0.075	0.052	0.00288
	Approximation	0.405	0.075	0.052	0.00288

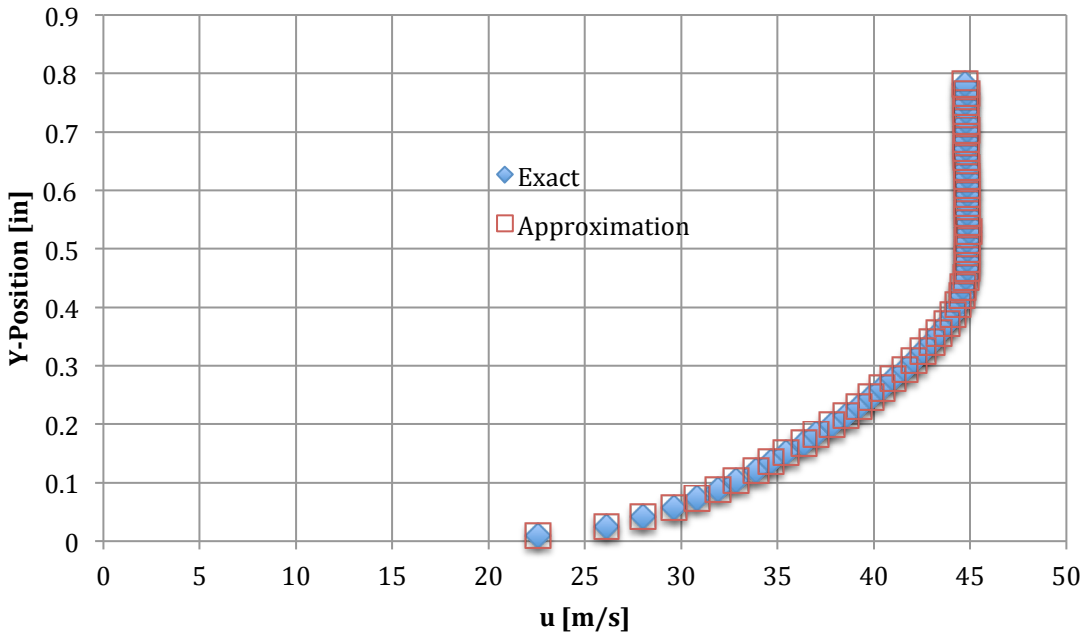


Figure 3.5.1. Turbulent boundary velocity profile using the exact calibration method and the approximated method for U = 45 m/s.

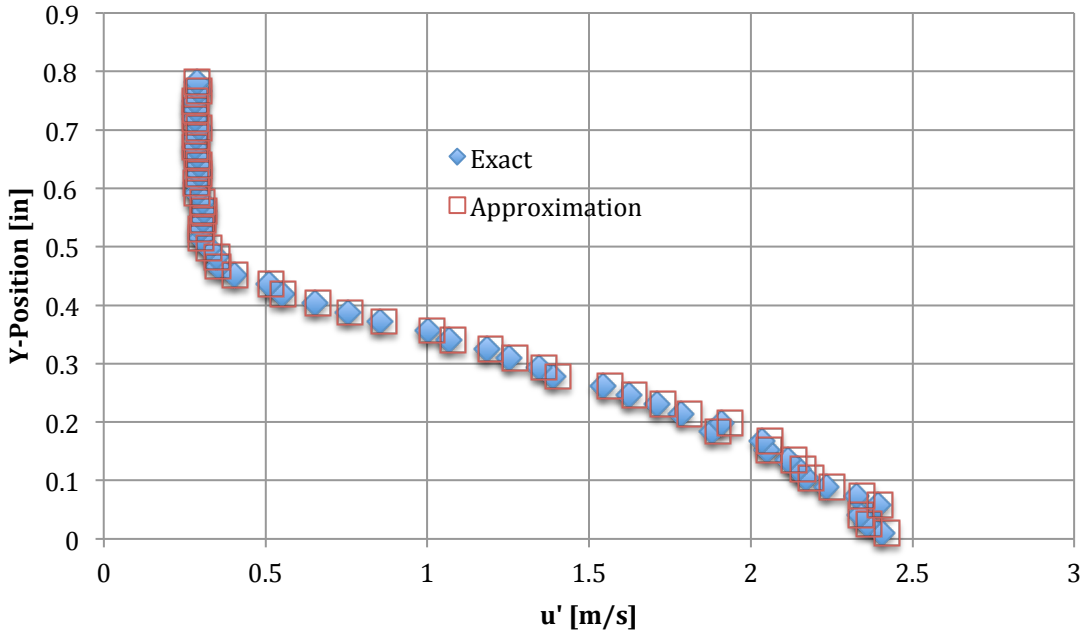


Figure 3.5.2. Fluctuating velocity of a turbulent boundary layer using the exact calibration method and the approximated method for U = 45 m/s.

From the results seen in Table 3.5.1 and Figure 3.5.1 above, the boundary layer data computed and the boundary layer velocity profile is exactly the same for all the different boundary layer characteristics. However, from Figure 3.5.2, the fluctuating velocity from using the approximate method yields a slightly larger fluctuating velocity inside the boundary layer with the greatest discrepancies seen in the middle of the boundary layer. This is seen consistently throughout the four freestream speeds tested. The largest percent error in the fluctuating velocity seen between the two methods for the $U_\infty = 45\, m/s$ case is 1.43%. Although there are discrepancies seen in the fluctuating velocity number using the approximation method, the percent error for the approximation method is small. In conclusion, the approximation method can be confidently used as a solution for the limited memory space on board the TFX–11v2.

4. CVA CALIBRATION METHOD

4.1. Traditional Hot-Wire Calibration Method

A hot-wire anemometer can be used to determine the velocity of a flow by relating the amount of heat transferred from the hot-wire into the environment to the flow velocity around the hot-wire. For the research work done for the rest of this thesis, I_w, the current necessary to keep the wire voltage, V_w, constant across the hot-wire, will be related to the velocity of the flow, u. This is different from the work done in Chapter 2, where the voltage V_o is used to relate to the velocity of the flow. This is done because the BLDS – CVA does not have the ability to output V_o as mentioned in Section 3.2. Note that I_w is measured as a voltage that is proportional to the current across the hot-wire. However, this does not create any discrepancies in the work done throughout this thesis as both parameters can be used to relate to the velocity of the flow. This is true through the relationship in Equation (4.1.1),

$$V_o = \left(1 + \frac{R_2}{R_F} + \frac{R_2}{R_w + R_L}\right)\frac{I_w \times (R_w + R_L)}{2}. \tag{4.1.1}$$

The I_w voltage needed across the hot-wire required to keep the voltage, V_w, across the wire constant can be related to the velocity of the flow, u, by using a simple inverted calibration that follows the power law in the form of [10],

$$u = P + QE^k. \tag{4.1.2}$$

The variable E is the mean voltage output proportional to the current through the hot-wire from a given set of measured calibration data. P, Q, and k are constants determined by MATLAB's least squares curve fit solver. The mean square error (MSE) is used to determine the goodness of fit of the power law for the calibration data and can be calculated using Equation (4.1.3),

$$MSE = \frac{1}{N} \sum \left(u_{fit} - u_{meas}\right)^2. \tag{4.1.3}$$

A sample of the results of a power law fit to calibration data for a velocity range of $u = 5 - 50\ m/s$ using $V_w = 0.69V$ and the probe described in Section 2.1 can be seen in Figure 4.1.1. The reason for using a power law instead of other functions, such as a polynomial, is mainly due to the fact that the heat transfer law that governs the hot-wire is in the form of a power law, see Equation (4.2.2). Other reasons include, the power law allows for plausible extrapolation of the calibration function outside of the velocity range over which the hot-wire was calibrated, and it employs a relatively small number of constants, 3 – or just two if the power is taken as fixed. The latter feature allows for the possibility of developing a calibration function using as few as 3 or perhaps only 2 calibration data points. If, say, a fourth order polynomial was used instead of the power law, there would be five constants and there would be a need for at least five calibration data points.

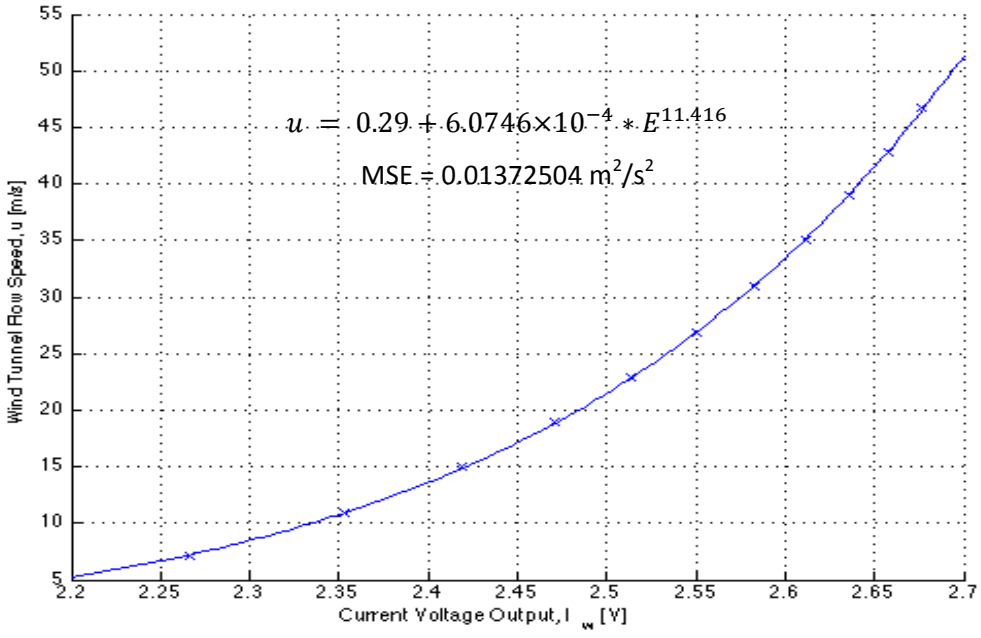


Figure 4.1.1. Inverted calibration curve for voltage output proportional to current with power law fit.

4.2. Effect of Temperature Drift

Although a power law curve fit works very well with the hot-wire calibration, there is a problem that arises with the of the calibration curve when temperature drift is present during the calibration. Calibrations for a hot-wire are only accurate if the environmental conditions (temperature, pressure) remain unchanged from the beginning of the calibration to the end. These same conditions must also prevail in the flow to be measured. If the amount of heat-transfer changes for the same velocity due to changes in the ambient conditions, such as the density and temperature, the calibration curve must be altered to correct fur such changes. Usually, a whole new calibration curve is constructed for the new ambient conditions. This phenomenon can be seen mathematically in the hot-wire models developed by Collis Williams in Equations (4.2.1)-(4.2.3) [14],

$$h = \frac{Nu\, k_f}{D} \tag{4.2.1}$$

$$Nu = \left(0.24 + 0.56\, Re_f^{0.45}\right)\left(\frac{T_f}{T_\infty}\right)^{0.17} \tag{4.2.2}$$

$$Re_f = \frac{\rho_f\, u\, D}{\mu_f}. \tag{4.2.3}$$

This can also be seen visually in Figure 4.2.1 when MATLAB tries to least squares curve fit the power law to a set of calibration data that has temperature drift. This specific set of calibration data, taken over a velocity range of $u = 10 - 60\, m/s$ in the Northrop Grumman Research Wind Tunnel with $V_w = 0.69V$ and the hot-wire described in Section 2.1, included a temperature drift from 19.9℃ to 25.9℃. This temperature drift is due to the heat generated by the fan in the closed-circuit wind tunnel used to do the experiment.

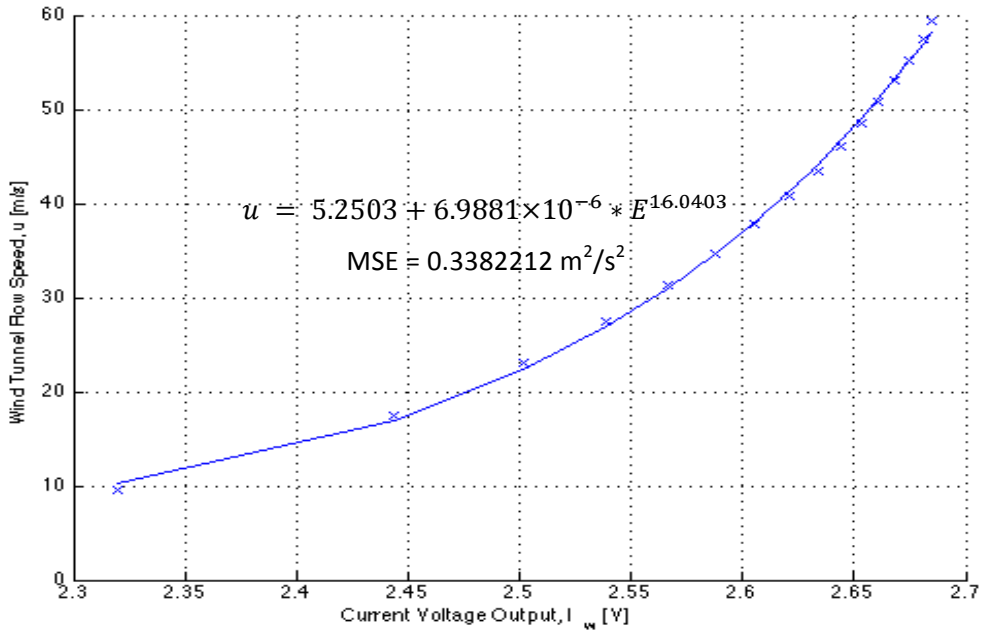


Figure 4.2.1. Least squares fitting a power law function to a calibration data set with temperature drift.

4.3. Solution for Temperature Drift

In order to get an accurate calibration curve, where all the data points are referenced to a single temperature value, data processing of the calibration data is

required. From the BLDS – CVA offload data file, the freestream flow velocity can be obtained using Equation (4.3.2), the ambient temperature can be obtained directly, and the ambient pressure can be obtained using a single point calibration.

$$\rho = 1.2929 \frac{kg}{m^3}\left(\frac{273.15\ K}{T_\infty}\right)\left(\frac{P_\infty}{760\ mm\ Hg}\right) \tag{4.3.1}$$

$$u = \sqrt{\frac{2\left(\Delta V_p\right)}{\rho}} \tag{4.3.2}$$

Using the velocity calculated from Equation (4.3.2), the ambient temperature, and ambient pressure at each calibration point, these values will be part of the input for the Engineering Equation Solver (EES) model developed previously by Will Neumeister [10] and used to compute the theoretical output voltage, I_w. Then with the same velocity, the ambient temperature and ambient pressure will be changed to the referenced conditions and see what the theoretical output voltage, I_w, would be without the temperature drift. The difference in I_w due to the temperature drifts is then applied to the I_w of the measured data to correct for the temperature drift seen during the calibration.

The result from applying this method to the data set shown in Section 4.2 can be seen below in Figure 4.3.1. This shows the need to correct for the temperature drift during the hot-wire calibration as the MSE was reduced to about $\frac{1}{7}$ of the uncorrected temperature drift MSE.

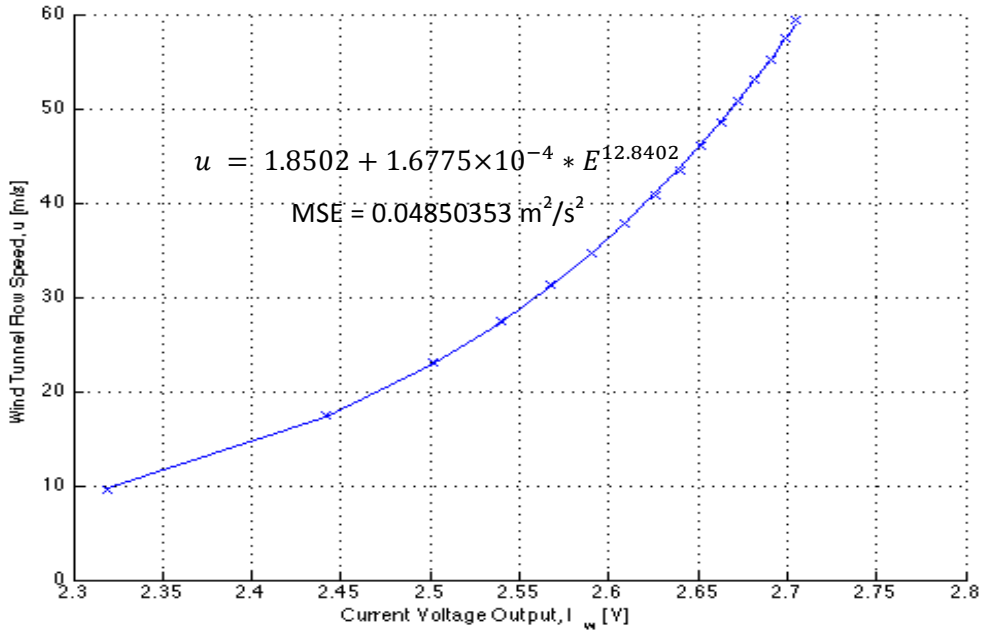


Figure 4.3.1. Corrected calibration curve where temperature drift was seen during the calibration of the hot-wire anemometer.

4.4. Autonomous Calibration for Any Ambient Conditions

As mentioned before, a problem with using the BLDS – CVA in flight is that there is no way to calibrate the hot-wire in flight. It also appears impractical to calibrate the hot-wire for all possible flight conditions on the ground because this would require reproducing flight conditions in a laboratory setting. There needs to be a solution to calibrate the hot-wire for the flight conditions in order for the BLDS – CVA to be a viable device to be used for flight testing. The solution proposed here requires that the exponent, k, in the power law calibration function be held constant and by relating the constants, P and Q, using a single freestream data point along with the temperature and pressure at the corresponding ambient conditions.

Using the thermal/electric model created by Will Neumeister in EES [10], the theoretical values at various ambient conditions were tested. The test includes the five

conditions listed in Table 4.4.1. The V_w is set for all these conditions so that the $OHR \approx 2.0$ at $u = 7.0\ m/s$. The hot-wire modeled is the same as the one described in Section 2.1.

Table 4.4.1. Conditions used to test the autonomous calibration method.

	V_w [V]	Pressure [kPa]	Temperature [°C]
Sea Level	0.65	101.325	20
60,000 feet	0.38	7.171	-56.5
0°C	0.62	101.325	0
40°C	0.65	101.325	40
40 kPa	0.55	40	20

To see if keeping the power law constant is a reasonable solution for the problem, a power law curve fit with a constant exponent was implemented for all five sets of data. From Figure 4.4.1, it can be seen that a constant exponent in the power law can be adapted to a wide range of ambient conditions and still yield an accurate inverted calibration curve given the correct constants, P and Q. The mean square error (MSE) for each case across a wide variety of exponent power values can be seen in Figure 4.4.2. From this study, $k = 10$ would yield a power law inverted calibration curve that will minimize the MSE seen across various possible ambient conditions.

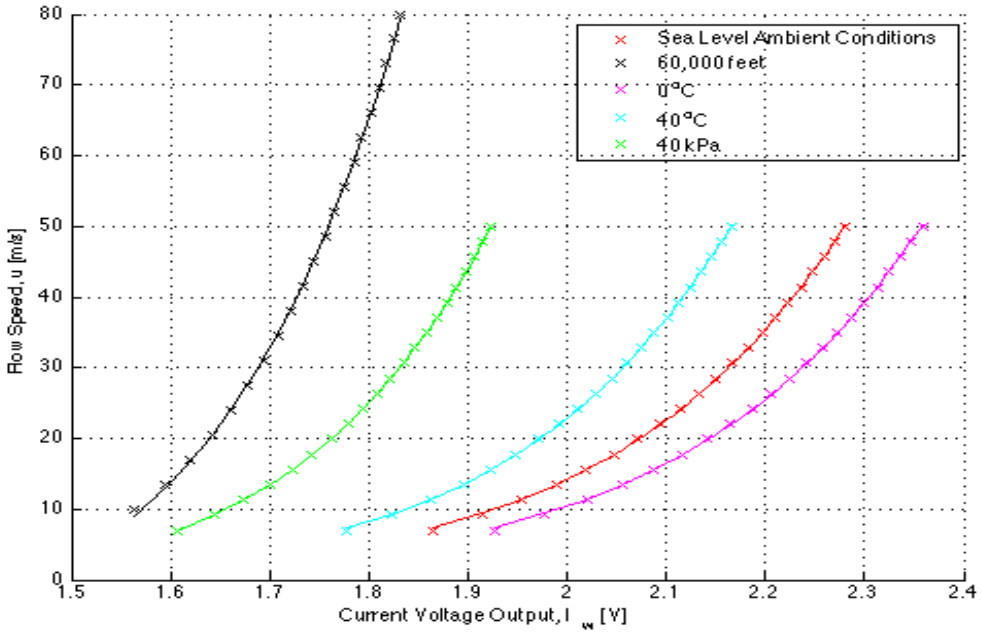


Figure 4.4.1. Inverted theoretical calibration curves with k = 10 for a variety of ambient conditions.

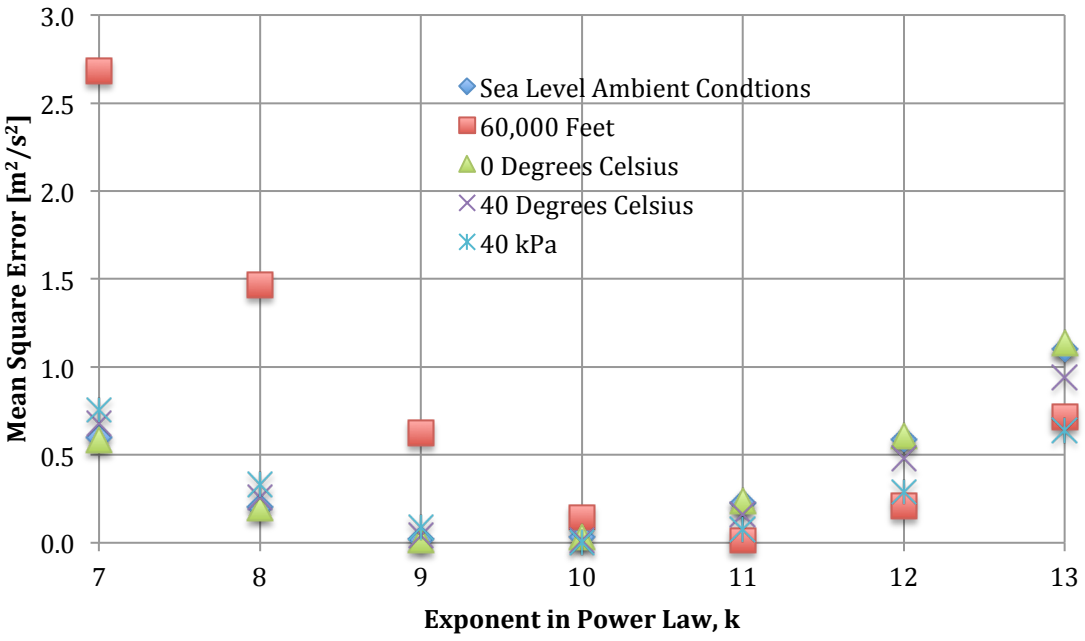


Figure 4.4.2. Mean square error for the different ambient conditions over a range of constant exponent, k, in power law.

Now that the exponent in the power law is set to be constant at $k = 10$, there are still two unknowns and only one freestream data point. In order for this to work, there

needs to be a way to relate the constants, P and Q. From visual inspection of the trend between P and Q, it was seen that they are, approximately, linearly related to each other. Fitting a line to the trend in MATLAB using the *polyfit* function yields the equation,

$$Q = -0.0197P + 0.0320. \tag{4.4.1}$$

The equation of the line has a MSE of $3.128\times10^{-5}\left(\frac{m/s}{V^{10}}\right)^2$. The MSE here is used to determine the goodness of fit of the linear fit for the numerically determined Q and P constants. This can be seen in Figure 4.4.3.

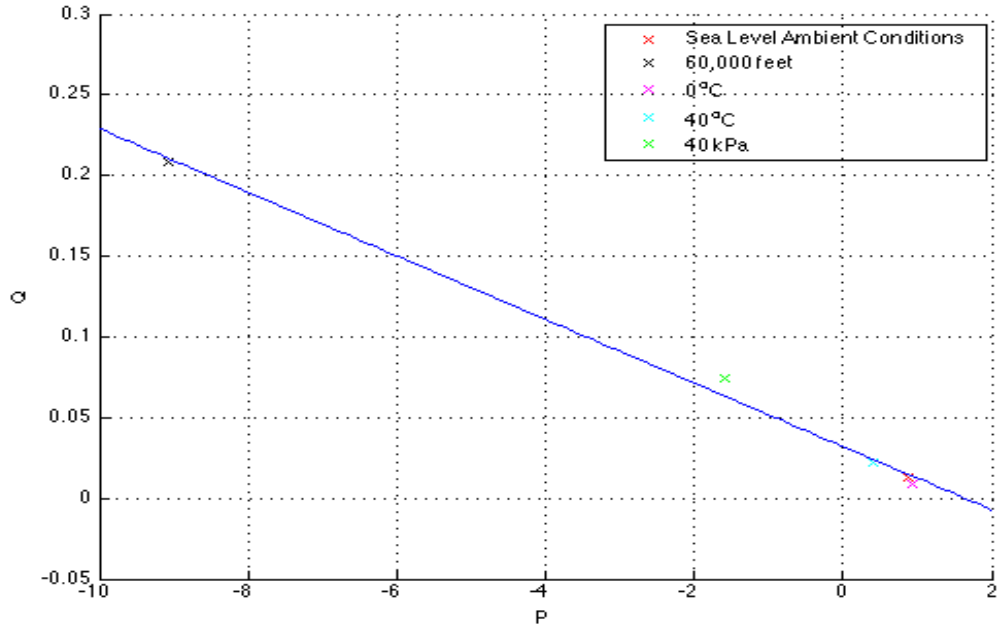


Figure 4.4.3. Relationship between constants P and Q from power law for various ambient conditions.

Now that there are two Equations, (4.1.2) and (4.4.1), five variables, u, P, Q, E, and k, and three knowns, P, Q, and k, the system of equations can be solved to give a calibration function. The calibration function from this method is as follows,

$$u = \frac{U_\infty - 0.0320E_\infty^{10}}{1 - 0.0197E_\infty^{10}} + \left[0.0320 - 0.0197\left(\frac{U_\infty - 0.0320E_\infty^{10}}{1 - 0.0197E_\infty^{10}}\right)\right]E^{10}. \quad (4.4.2)$$

To evaluate the accuracy of the resulting calibration function, the calibration function is applied to the five data sets describe in Table 4.4.1 above. In Table 4.4.2, the MSE between the calibration function and the data sets from EES can be seen and a sample calibration curve using Equation (4.4.2) can be seen in Figure 4.4.4. It can be verified that this method can theoretically calibrate a hot-wire with a single data point along with the ambient conditions at that data point without performing a full calibration.

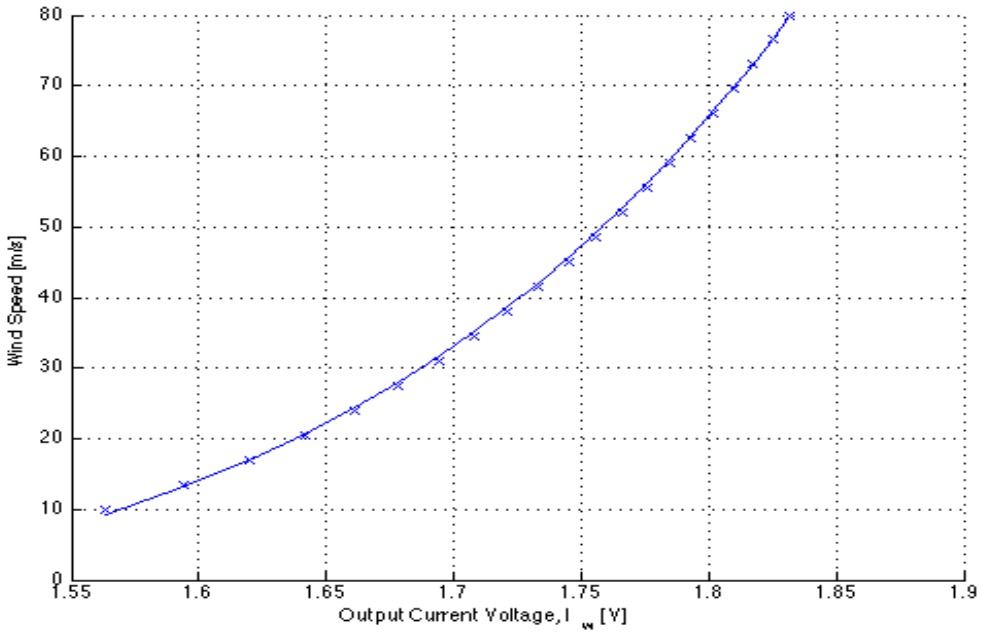


Figure 4.4.4. Theoretical calibration curve using autonomous method for ambient conditions at 60,000 ft.

Table 4.4.2. MSE of the theoretical calibration function at various ambient conditions.

	Sea Level	60,000 Feet	0 °C	40 °C	40 kPa
MSE $[\frac{m^2}{s^2}]$	0.0424	0.2808	0.0568	0.0192	0.0826

Although this method is a valid theoretical alternative to doing a full calibration at the ambient conditions that the hot-wire will be used in, it also has to be proven to work experimentally as well. The theoretical study described from above used Collis and Williams' Nusselt Number correlation to predict the heat transfer between the hot-wire and the ambient environment [14]. However, in their study there are some simplifications made to their research. This includes but is not limited to a two-dimensional analysis and it only considers convection between the hot-wire and the ambient environment and not any conduction lost through the prongs that hold the hot-wire. Nor does it account for heat loss through radiation. As was seen in Will Neumeister's thesis, Collis and Williams' model of the hot-wire heat transfer laws, always under-predicts the amount of heat transfer that actually occurs [10]. In addition, the amount that their model under-predicts the heat transfer laws is not consistent. It under-predicts the amount of heat loss at $u \approx 7\ m/s$ by 10.7% for a $V_w = 0.69V$ and at $u \approx 46\ m/s$ by 7.1% for a $V_w = 0.69V$. The theoretical and experimental calibration curves are plotted on top of each other in Figure 4.4.5 to show the difference in the offset between the low and high velocities.

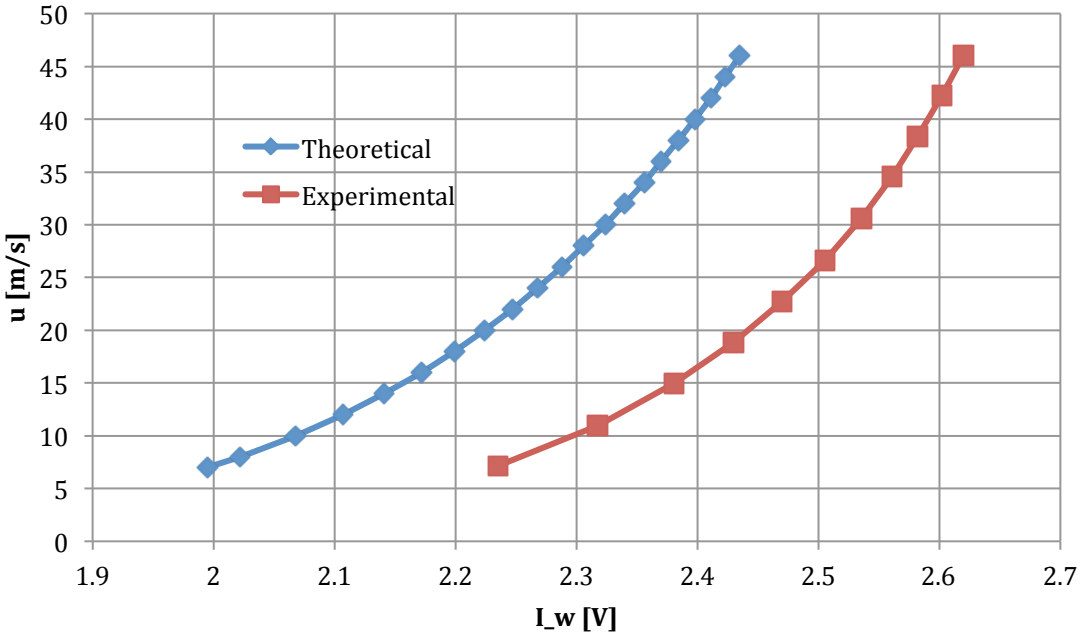


Figure 4.4.5. Offset of the calibration curve between the theoretical and experimental data sets at sea level ambient conditions.

Since the theoretical and experimental data is not offset by a simple shift, a similar study was conducted to verify that the calibration function can be applied to the calibration data collected from the BLDS-CVA. Four sets of calibration data with different wire voltages settings were collected experimentally using the BLDS – CVA in Cal Poly's Mechanical Engineering 2x2 foot wind tunnel. The data sets were collected with $V_w = 0.45V, 0.55V, 0.65V,$ and $0.69V$ at sea level ambient conditions. Note that this means the lower wire voltage settings do not reach the desired $OHR \approx 2$ at low velocities for maximum sensitivity.

Setting the exponent, k, constant in the power law for various values shows that there is an accurate solution possible. An analysis was completed to pick the best k value in order to fit a wide variety of wire voltage settings. To pick the best k value for the BLDS – CVA application, the corresponding MSE has to be minimized. The results from

this analysis can be seen in Figure 4.4.6. As can be seen, setting $k = 13$ and solving for the best P and Q for each of the cases yields the best results and can be seen in Figure 4.4.7.

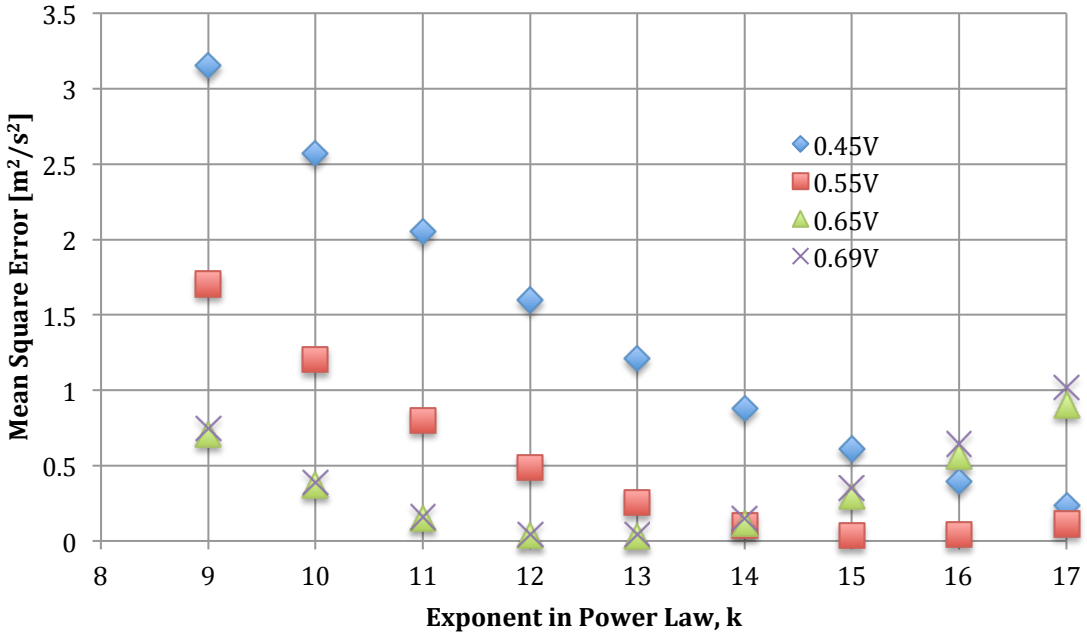


Figure 4.4.6. MSE for the different wire voltage settings across various values of the constant, k, in the power law.

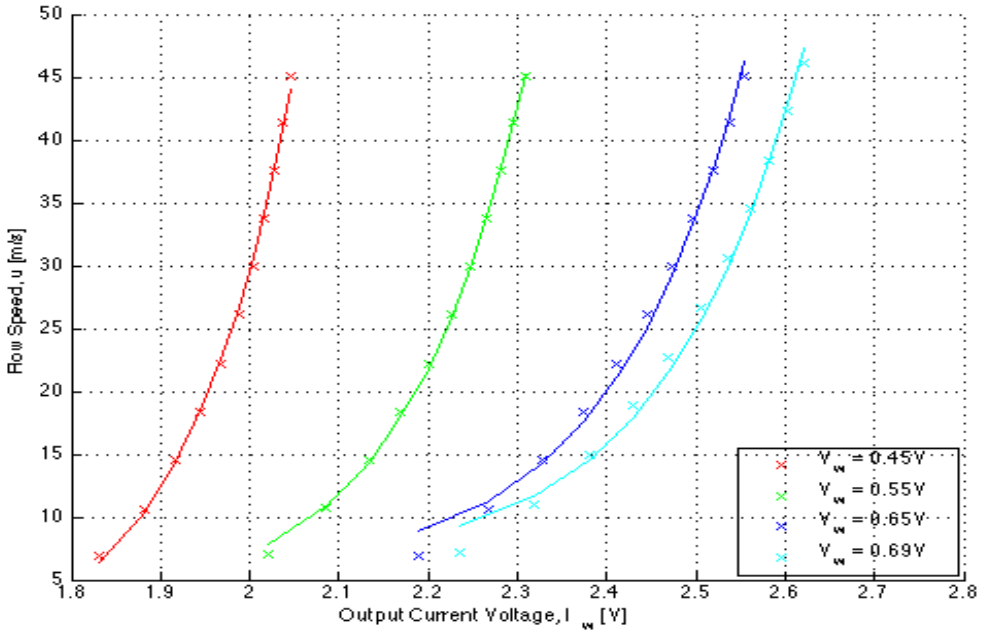


Figure 4.4.7. Inverted experimental calibration data for k = 13 for a variety of wire voltage settings.

It should be noted that when $k = 15$, a better MSE average for the four calibration data sets is seen. However, from the same analysis as done previously with the theoretical data set, there is believed to be a correlation between the k value and the OHR. The lower the OHR, the higher the k value needs to be in order for an accurate curve to be fitted to the data set. This is visually present in Figure 4.4.7. For $V_w = 0.45V$ and $0.55V$, the highest OHR seen by the hot-wire is around 1.7 and 1.8 respectively at $u = 7\ m/s$. Where as for the $V_w = 0.65V$ and $0.69V$ case, the OHR is about 2.0 at $u = 7\ m/s$. For the two lower wire voltage settings the power law fit did not capture the lowest velocity and highest velocity as well as the two higher wire voltage settings. Since an $OHR \approx 2$ is always desired for better sensitivity and would be used for all applications of the BLDS – CVA, the value of k with the lowest average MSE, minus the $V_w = 0.45V$ calibration data, was chosen. The k value that corresponds to the lowest average MSE minus the $V_w = 0.45V$ calibration data is $k = 13$.

With the k chosen in the power law, there are still two unknowns, P and Q, to relate to a single data point. A similar analysis, as completed with the theoretical calibration data, was done for the experimental data. The result can be seen in Figure 4.4.8. A linear trend can be seen from the data points and by using MATLAB's *polyfit* function. Equation (4.4.3) can relate P and Q given a single calibration point,

$$Q = -5.02\times10^{-4}P + 7.98\times10^{-4}. \tag{4.4.3}$$

The MSE for the curve fit is $2.3959\times10^{-7}\left(\frac{m/s}{V^{10}}\right)^2$. The MSE here is used to determine the goodness of fit of the linear fit for the numerically determined Q and P constants. This can be seen in Figure 4.4.8.

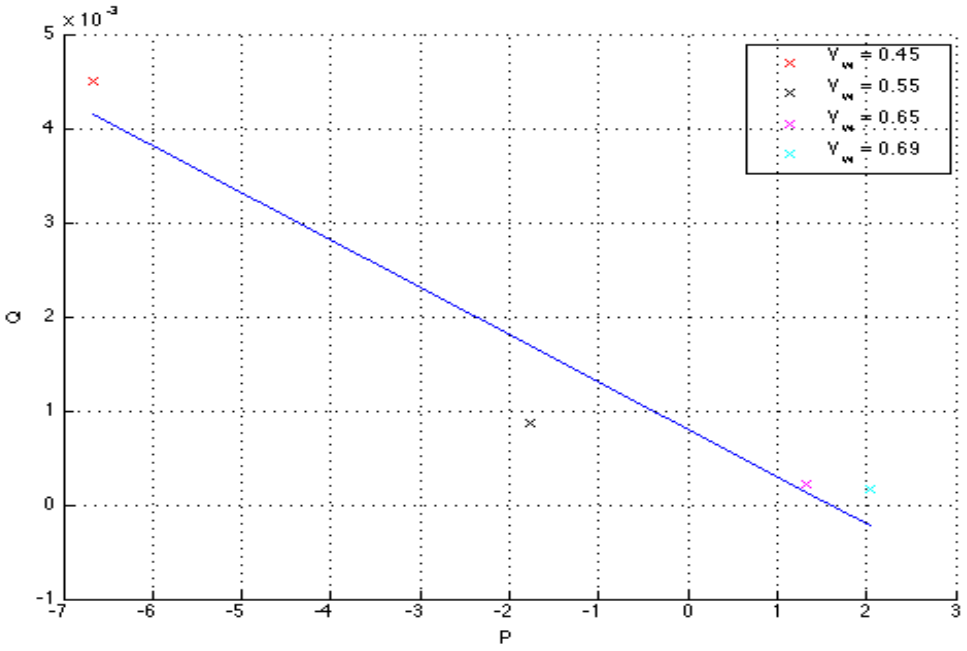


Figure 4.4.8. Relationship between P and Q constants from the power law for the various wire voltage settings.

Combining Equation (4.1.2) and (4.4.3) with a single freestream data point yields,

$$u = \frac{U_\infty - 7.98\times10^{-4}E_\infty^{13}}{1 - 5.02\times10^{-4}E_\infty^{13}} \ldots$$

$$+ \left[7.98\times10^{-4} \ldots \right. \tag{4.4.4}$$

$$\left. - 5.02\times10^{-4}\left(\frac{U_\infty - 7.98\times10^{-4}E_\infty^{13}}{1 - 5.02\times10^{-4}E_\infty^{13}}\right)\right] E^{13}.$$

Using (4.4.4) with the experimental calibration data sets of all four different wire voltage settings yields the MSE shown in Table 4.4.3. A sample calibration curve using this method can be seen in Figure 4.4.9.

Table 4.4.3. MSE of the experimental calibration function at various wire voltage settings.

	0.45V	0.55V	0.65V	0.69V
MSE $[\frac{m^2}{s^2}]$	2.6157	1.8306	0.0340	0.1978

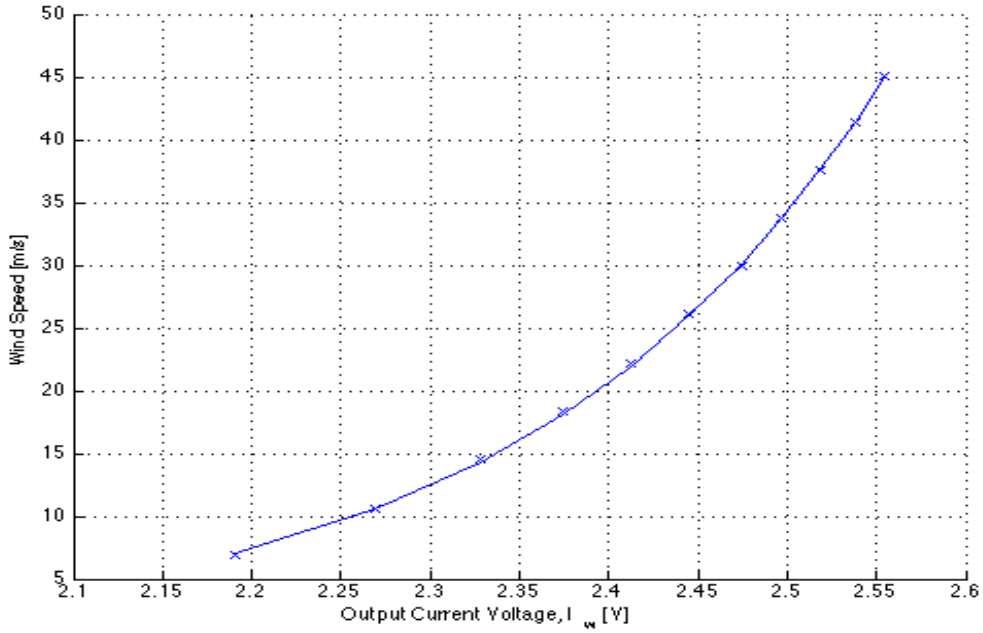


Figure 4.4.9. Experimental calibration curve using autonomous calibration method for V_w = 0.65V.

Although a high MSE is seen for the lower wire voltage settings, this will not pose a problem when implementing this method for the BLDS – CVA because a low wire voltage setting, and therefore low OHR, will not be used during regular operation of the

BLDS – CVA. Note that the way this method is developed, the calibration curve is based on the ambient conditions at the freestream point and therefore assumes that any measurements made which employ this calibration will be done at essentially the same ambient conditions.

While the solution presented here for calibrating a hot-wire autonomously is valid for most cases, it does not always work. For example, a previous set of calibration data taken at sea level ambient conditions to understand the effect of a low OHR was used to test the calibration function (4.4.4). The OHR seen from this calibration data set is lower than generally desired: $OHR \cong 1.31 \sim 1.51$. Using the calibration data from this test did not produce a result, as explained further below. Adapting the calibration function for this set of data yields the following calibration function,

$$u = 47.406 - 0.023E^{13}. \tag{4.4.5}$$

The problem with the calibration function (4.4.5) is that the higher the input E is the lower the output u is. This can be seen in Figure 4.4.10. However, this should not be the case because it is known that the flow velocity should increase as the input voltage E increased because more cooling occurs at higher flow velocities. Although this shows up at low OHRs this is not a serious problem because, again, a low OHR will not be used during the operation of the BLDS-CVA. The V_w will be picked so that the OHR will be high and in the range of $1.8 - 2.0$. Even though this is not an immediate problem and does not hinder the progress of adapting this method to the BLDS-CVA, it should be noted that the calibration function does not represent the actual physics of the hot-wire over a very wide range of ambient and operating conditions. If the proposed function (4.4.4) did represent the actual physics of the hot-wire, any single data point used in the

calibration function should yield a physically possible result where u increases as E increases.

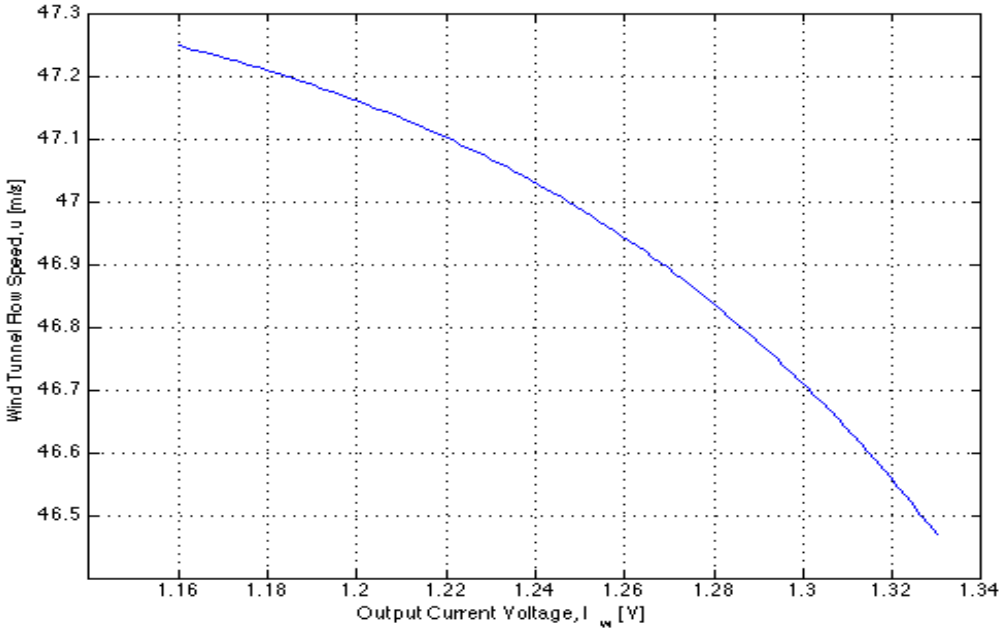


Figure 4.4.10. Calibration function when the OHR is 1.31-1.51 for u = 7 m/s to 46 m/s.

4.5. Accuracy of Autonomous Calibration Method

Using the autonomous calibration method described in Section 4.4 is only an attempt to come up with a calibration curve without performing a traditional calibration. As a result, the results from calculating the boundary layer characteristic using the autonomous calibration method will not yield as accurate results as the boundary layer characteristics calculated using the traditional calibration method. However, it does provide a solution to forming a calibration curve when a traditional calibration cannot be performed. The question now becomes: how accurate is this alternative solution, and, is it reliable enough to provide accurate boundary layer measurements? To make the comparison, a turbulent boundary layer velocity profile was measured by the BLDS – CVA and the two calibration methods were used to calculate the boundary layer mean

and fluctuating velocities. The experimental setup is similar to the one described in Section 2.1 except the hot-wire probe on the BLDS – CVA is placed at $x = 23.4\ inches$.

The boundary layer velocity profiles and boundary layer fluctuating velocity profiles at $U_\infty = 23.3\ m/s, U_\infty = 30.9\ m/s, U_\infty = 39.2\ m/s,$ and $U_\infty = 47.5\ m/s$ collected by the BLDS – CVA using both the traditional calibration method and the autonomous calibration method can be seen in Figure 4.5.1 - Figure 4.5.4 and Figure 4.5.5 - Figure 4.5.8, respectively. In the boundary layer velocity profiles, it can be seen that the results from the two methods yields different velocities for most y – locations with a consistent trend of the autonomous calibration method under-predicting the velocity data in the boundary layer velocity profile data. In the boundary layer fluctuating velocity profiles, the autonomous calibration method tends to over-predict the intensity of the fluctuations except for select data points near the flat plate surface at the lowest speed.

The results are very similar and more comparable however when comparing the boundary layer characteristics calculated from both methods. These values can be seen in Table 4.5.1.

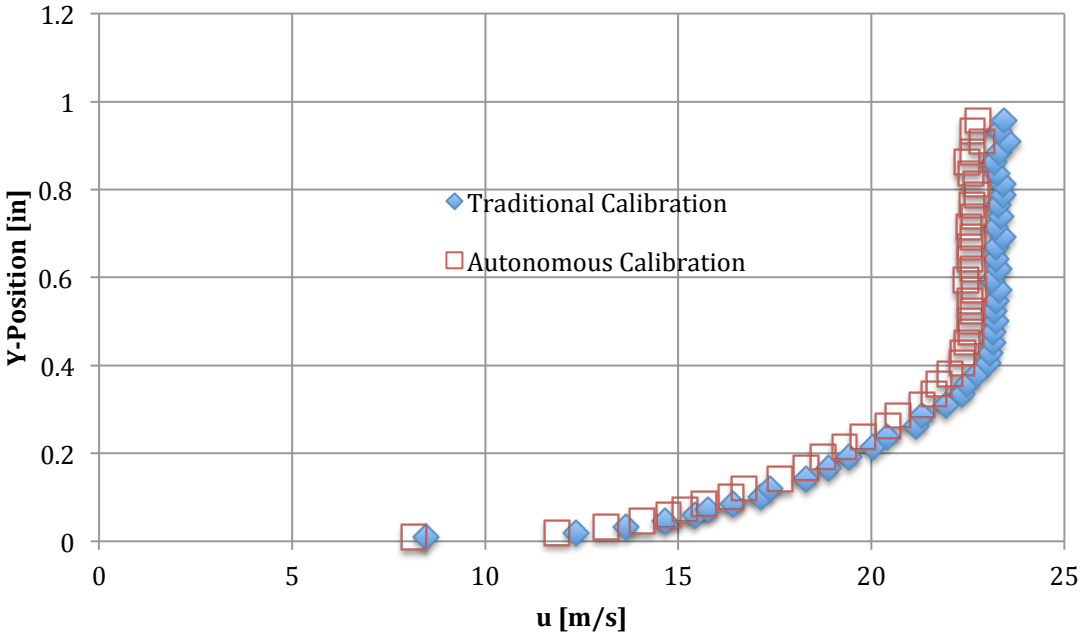


Figure 4.5.1. Boundary layer velocity profile captured by BLDS - CVA using the traditional calibration and autonomous calibration at U = 23.3 m/s.

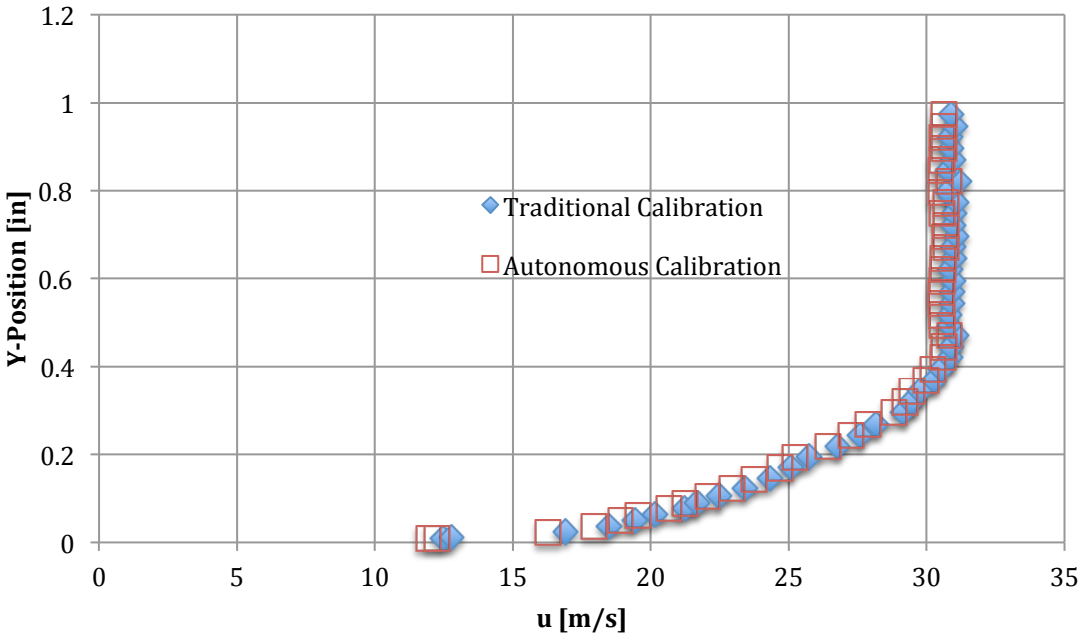


Figure 4.5.2. Boundary layer velocity profile captured by BLDS - CVA using the traditional calibration and autonomous calibration at U = 30.9 m/s.

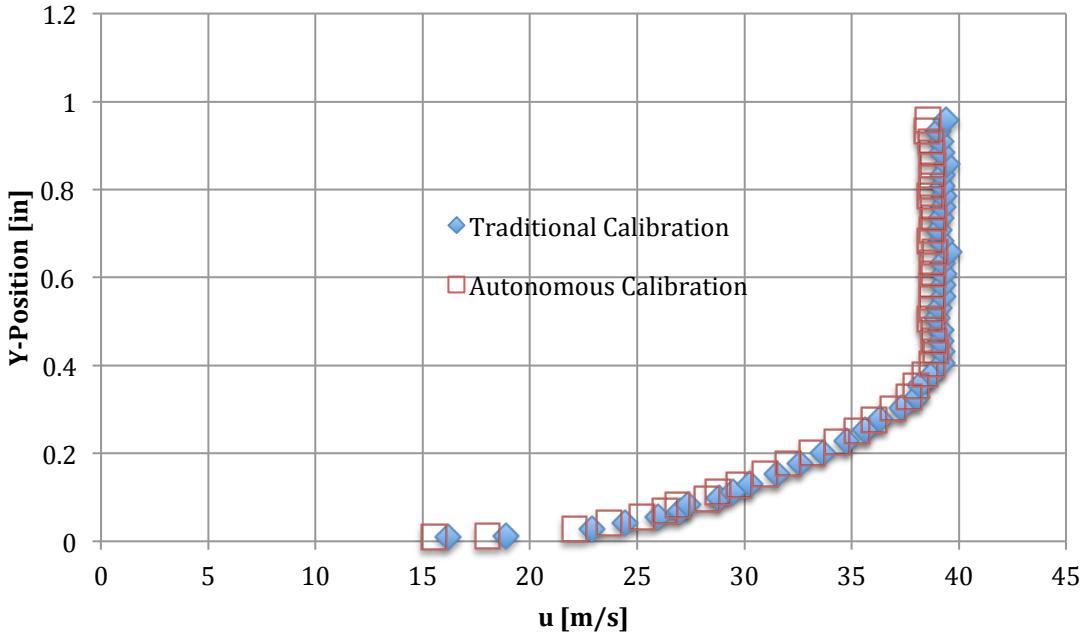


Figure 4.5.3. Boundary layer velocity profile captured by BLDS - CVA using the traditional calibration and autonomous calibration at U = 39.2 m/s.

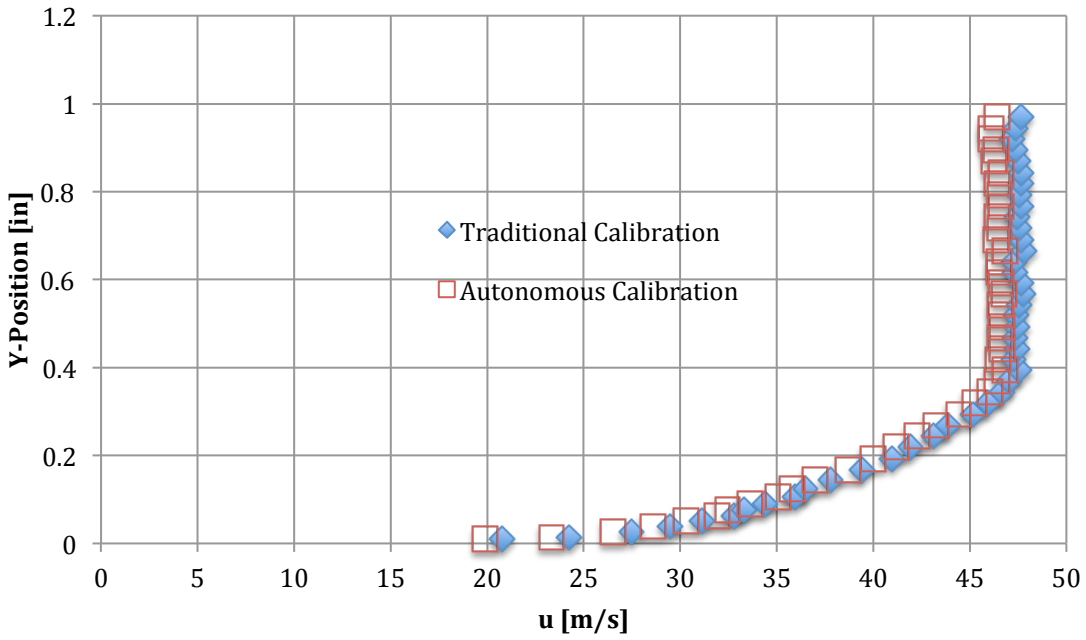


Figure 4.5.4. Boundary layer velocity profile captured by BLDS - CVA using the traditional calibration and autonomous calibration at U = 47.5 m/s.

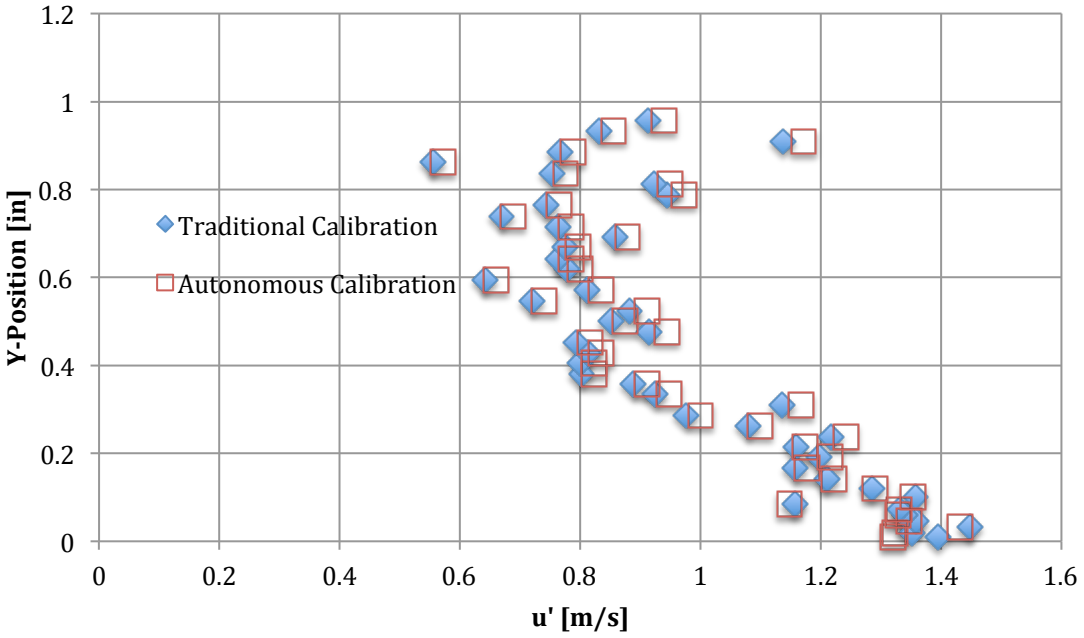


Figure 4.5.5. Boundary layer fluctuating velocity profile captured by BLDS - CVA using the traditional calibration and autonomous calibration at U = 23.3 m/s.

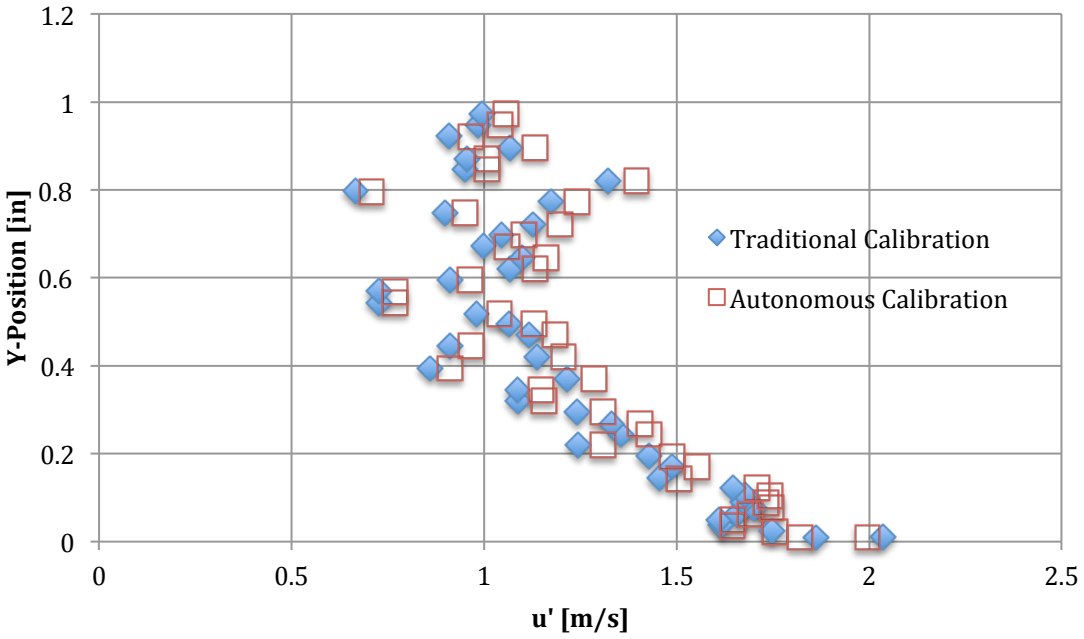


Figure 4.5.6. Boundary layer fluctuating velocity profile captured by BLDS - CVA using the traditional calibration and autonomous calibration at U = 30.9 m/s.

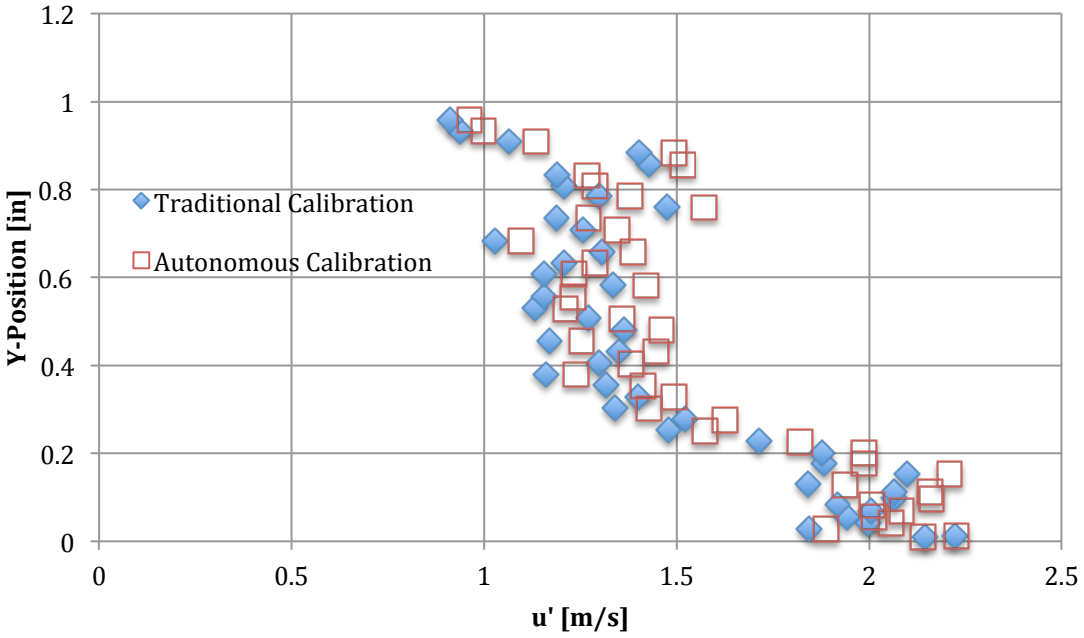


Figure 4.5.7. Boundary layer fluctuating velocity profile captured by BLDS - CVA using the traditional calibration and autonomous calibration at U = 39.2 m/s.

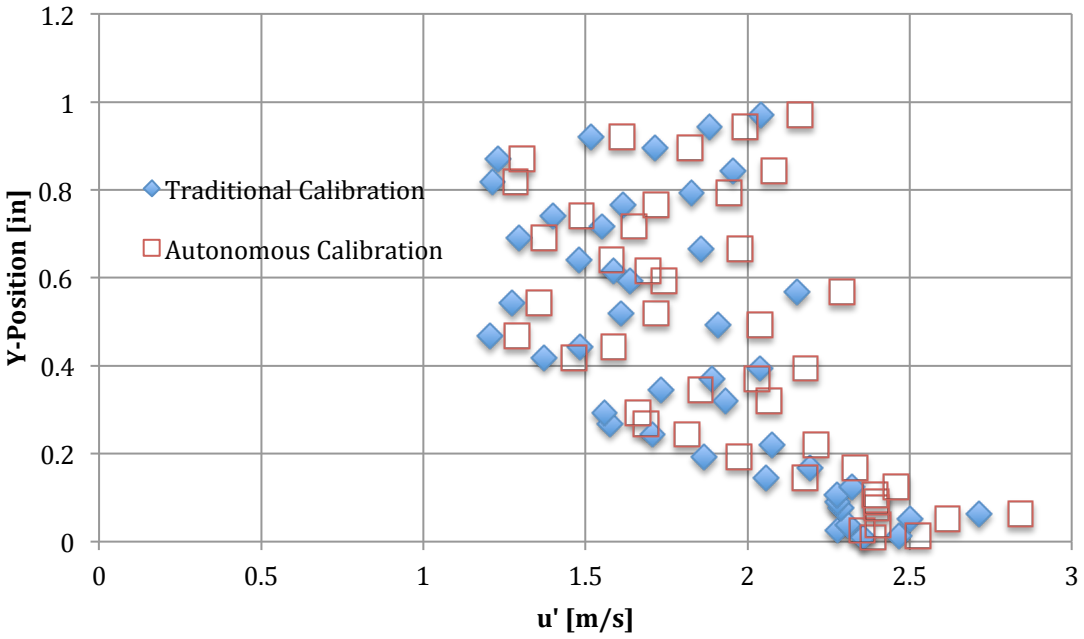


Figure 4.5.8. Boundary layer fluctuating velocity profile captured by BLDS - CVA using the traditional calibration and autonomous calibration at U = 47.5 m/s.

Table 4.5.1. Turbulent boundary layer characteristics comparison using both the traditional and autonomous calibration methods.

U_∞	Method	δ (99%) [in.]	δ^* [in.]	θ [in.]	C_f
23.3 m/s	Traditional	0.4305	0.0785	0.0526	0.00332
22.6 m/s	Autonomous	0.4326	0.0800	0.0532	0.00331
30.9 m/s	Traditional	0.4033	0.0776	0.0515	0.00301
30.6 m/s	Autonomous	0.3999	0.0790	0.0518	0.00292
39.2 m/s	Traditional	0.3832	0.0728	0.0488	0.00289
38.7 m/s	Autonomous	0.3747	0.0734	0.0485	0.00284
47.5 m/s	Traditional	0.3694	0.0698	0.0470	0.00278
46.5 m/s	Autonomous	0.3450	0.0694	0.0462	0.00277

It should be noted that a similar temperature drift correction was applied to the traditional calibration data using the method explained in Section 2.2, the only difference being the exact temperature drift is known for each data point so the linear temperature change assumption with time is not needed. However, even with the temperature drift correction, it was still not able to correct for the large temperature drift seen during the experiment due to the non-linear relationship between the voltage change and ambient temperature change.

From these results, the smallest percent difference that was seen in the data when using the autonomous calibration method instead of the traditional calibration method was 4.2% at the second lowest speed setting and the largest percent difference was 4.6% at the lowest speed setting for the mean velocity data. For the fluctuating velocity data, the smallest percent difference seen was 3.3% at the lowest speed setting and 7.2% at the highest speed setting. For the boundary layer thickness, the smallest percent difference seen was 0.4% at the lowest speed setting and the largest percent difference was 6.6% at the highest speed setting. For the displacement thickness, the smallest percent difference seen was 0.6% at the highest speed setting and the largest percent difference was 1.9% at the lowest speed setting. For momentum thickness, the smallest percent difference seen

was 0.6% at the second lowest speed setting and the largest percent difference seen was 1.7% at the highest speed setting. For local coefficient of skin friction, the smallest percent difference seen was 0.3% at the lowest speed setting and the largest percent difference seen was 3.0% at the second lowest speed setting. Based on these results, there is no consistent deviation found between the two methods from one flow speed to another. However it is noticeable that the autonomous method consistently over-predicts the displacement thickness and the momentum thickness at high speeds and under-predicts the displacement thickness and the momentum thickness at lower speeds. The autonomous method also tends to under-predicts the local coefficient of skin friction and the boundary layer thickness except for the lowest speed setting.

Although the error associated with using the autonomous calibration method instead of the traditional calibration method cannot be avoided, this method can still be used confidently to calculate boundary layer characteristics from the boundary layer measurements made by the BLDS – CVA due to the relatively low maximum percent error of 6.6%. This 6.6% also includes the error due to the large temperature shift that cannot be accounted for even with the correction. However, this will not pose a problem as temperature drifts are not expected during the BLDS – CVA operation in flight.

5. CONCLUSION AND RECOMMENDATIONS

The desire to measure velocity fluctuations in addition to the average velocity in the boundary layer has led to the development of the BLDS – CVA. With the completion of this thesis, the challenges that are inherent with incorporating the CVA to the BLDS have been dealt with and the BLDS – CVA is now ready for flight test applications. The CVA system used to measure boundary layer velocity profiles was able to measure average boundary layer characteristics, such as the boundary layer thickness, displacement thickness, momentum thickness, and the local coefficient of skin friction, that are within 6.4% of the values measured by the traditional pressure probe. In addition, it was found that a portion of this error is due to temperature drift. Temperature drift is not expected during flight operation so it will not be a problem. However, if the BLDS – CVA will be used for wind tunnel application, a simple correction can be implemented for small temperature drift with the use of the CVA thermal/electrical computer model as demonstrated in Section 4.3. Also, the CVA system was proven capable of measuring a significant portion of the frequency spectrum of velocity fluctuations expected in a transitional or turbulent boundary layer when the hot-wire is operated at a high OHR, which is expected to be the case for all in flight applications.

Modifications were made to the BLDS to accommodate the CVA system. Pressure transducers with a smaller pressure range were used to increase the sensitivity for boundary layer measurements in the Cal Poly 2x2 wind tunnel and a new circuit board was designed and manufactured to accommodate the new CVA Daughterboard. The CVA Daughterboard has been designed and built to accommodate the CVA system onto the BLDS electronics and this device has been thoroughly tested to work on the

bench top and on the BLDS – CVA with the use of three new TFX–11v2 programs created specifically for the new hardware. The three programs allow for the testing and operation of the BLDS – CVA with either the CVA or with a traditional pressure probe. Modifications were done to the TFX–11v2 to allow for a better A/D reference that is needed for accurate velocity fluctuation measurements. The R27 0Ω resistor was removed and the regulated 5V power supply on board is used as the A/D reference. In addition, an approximate mathematical formulation was developed to address the limited memory space on board the TFX–11v2 and this provided positive results. All mean boundary layer characteristics, such as the boundary layer thickness, displacement thickness, momentum thickness, and the local coefficient of skin friction, calculated using the approximate mathematical formulation provided essentially identical values as the normal calibration method. The approximate method also gave good results for the fluctuation intensity; a maximum percent error of 1.43% for the $U_\infty = 45\ m/s$ was observed between the approximate and traditional method.

The need for a procedure to calibrate a hot-wire for ambient conditions in environments that cannot be reproduce in the lab has been addressed. An approximate calibration curve can be formed for any ambient condition as long as the freestream velocity and the corresponding I_w voltage can be measured. Using this method, errors of a few percent can be expected in the boundary layer mean characteristics values. Like the percent error observed during the evaluation of the hot-wire's accuracy in measuring boundary layer velocity profiles, a portion of the percent error is due to the temperature drift seen during the experiments in the wind tunnel. It should be noted that this method will not work at low OHRs. However, this should not pose a problem as a high OHR will

be used for all in flight applications to ensure that the BLDS – CVA can measure as much of the turbulence frequency spectrum as possible.

Even though a lot of work has been done to ensure the BLDS – CVA is flight ready, there is one final step that needs to be taken before flying the BLDS – CVA on an actual flight. As mentioned before, the pressure transducer needs to be replaced so the new pressure transducer can withstand the extreme environmental conditions experienced at high altitudes and has the ability to measure differential pressures at the high speeds expected from an aircraft. In addition, a new BLDS – CVA case needs to be designed and manufactured to enclose the larger, high-performance battery that is capable of withstanding the extreme environmental conditions at high altitudes and has enough current capacity to measure the desired number of boundary layer profiles.

Some items of interests that should be investigated in future work are the use of a smaller diameter hot-wire for better frequency response, performing a hot-wire calibration in a vacuum chamber to simulate high altitudes, and creating a device that can be used to prevent the hot-wire probe from coming into contact with the surface. Based on the results seen in Section 2.4, the need to capture higher turbulent fluctuation frequency is desired. The current diameter hot-wire used on the BLDS – CVA is not able to capture the turbulent fluctuations expected. With a smaller diameter hot-wire, this will be possible due to the frequency response of the hot-wire being a function of the hot-wire's diameter. However, it does come at the cost of the hot-wire's sensitivity. This would need to be addressed by application of offset and gain to the signal before the A/D converter.

Performing a hot-wire calibration in a vacuum chamber to simulate high altitudes would be a beneficial study to perform. Currently all the data used to validate the method described in Section 4.4 only consists of temperature change. Being able to calibrate a hot-wire and quantify the accuracy of the proposed method for lower ambient pressure would help better understand the boundary layer data that would be measured from flight testing. It was found that the AMETEK Rotron 90-AA3-126 fan would provide the performance and size requirement needed to perform a hot-wire calibration inside the vacuum chambers in the Cal Poly's Aerospace Engineering Department. The product specifications can be found in Appendix E.

One major concern in using the hot-wire on the BLDS is the possibility of the hot-wire breaking. There are two ways a hot-wire can break. One is by exceeding the hot-wire's operating temperature (resistance), and the other is by making contact with the delicate hot-wire itself. Not exceeding the hot-wire's operating resistance has been addressed through suggesting a V_w in the BLDS – CVA software. However, there is no current method implemented onto the BLDS – CVA to prevent the hot-wire from coming into contact with a surface. One possible solution for this problem is to utilize a specific port on the BLDS – CVA electronics that will shut of the stage if the circuit connected to the port becomes close. This allows for the option to create a metallic ring holder that can be fitted onto the hot-wire probe holder. When the ring holder comes into contact with a conducting surface, the circuit will be open and cause the stage to turn off. This stops the hot-wire from running into the surface. In addition, the dimension of the ring holder can be specifically designed so when the stage is shut off through this method the hot-wire distance from the surface is known and repeatable.

REFERENCES

[1] J. J Thibert, J. Reneaux, and R. V. Schmitt, "Maintaining Laminartiy by Boundary Layer Control," *ICAS proceedings*, 1990.

[2] R. V. Westphal, M. Bleazard, A. Drake, A. Bender, D. Frame, S. R. Jordan, "A Compact, Self-Contained System for Boundary Layer Measurements in Flight," AIAA-2006-3828, *AIAA Meeting Papers on Disc* [CD-Rom], Vol. 11, No. 10-13, AIAA, Reston, VA, 2006.

[3] A. Bender, R. V. Westphal, A. Drake, "Application of the Boundary Layer Data System on a Laminar Flow Swept Wing Model In-Flight," AIAA-2010-0373, *AIAA Meeting Papers on Disc* [CD-Rom], Vol. 15, AIAA, Reston, VA, 2010.

[4] J. L. Sproston, O. T. Goksel, "The Calibration of a Surface Static Tube," Aeronautical Journal, 1972, pp. 101-103.

[5] J. H. Preston, "The Determination of Turbulent Skin Friction by Means of Pitot Tubes," Journal of the Royal Aeronautical Society, vol. 58, no. 518, February 1954, pp. 109-121.

[6] R. V. Westphal, D. Frame, S. R. Jordan, A. Wanner Jr., B. Thompson, A. M. Bender, A. Drake "Design of a Third-Generation Boundary Layer Measurement System" AIAA, 2008.

[7] C. R. Ulk, "Implementation of a Conrad Probe on a Boundary Layer Measurement System," Cal Poly Digital Commons, 2010.

[8] R. V. Westphal, M. Prather, M. Toyooka, "RotableSingle-Hole Pressure Probe for Flow Velocity and Direction," AIAA-2004-2601, *AIAA Meeting Papers on Disc* [CD-ROM], Vol. 9, No. 9-10, AIAA, Reston VA, 2004.

[9] A. R. Wazzan, T. T. Okamura, and A.M.O Smith, "Spatial and Temporal Stability Charts for the Falkner-Skan Boundary Layer Profiles," Douglas Aircraft Company, 1968.

[10] W. D. Neumeister, "Hot-Wire Anemometer for the Boundary Layer Data System," California Polytechnic Statue University, San Luis Obispo, M.S. Thesis 2012.

[11] G.R. Sarma, "Flow Rate Measuring Apparatus," Patent No, 5074147, March 1990.

[12] G.R. Sarma, "Analysis of a Constant Voltage Anemometer Circuit," in *Instrument Measurement and Technology Conference*, Irvine, 1993, pp.731-736.

[13] G.R. Sarma, "Transfer function analysis of the constant voltage anemometer," *Review of Scientific Instruments*, vol. 69, no, 6, pp. 2385-2391, 1998.

[14] D. C. Collis and M. J. Williams, "Two-dimensional convection from heated wires at low Reynolds numbers," *Journal of Fluid Mechanics*, vol. 6, no. 3, pp. 357-384, 1959.

[15] P.S Klebanoff, "Characteristics of Turbulence in a Boundary Layer with Zero Pressure Gradient," NACA, 1247, 1955.

[16] D. Coles and E.A. Hirst, "*Computation of Turbulent Boundary Layers*", *1968 AFOSR-IFP-Stanford Con*., vol. II. Stanford, CA: Mechanical Engineering Department, Stanford University, 1968.

[17] W.H. Beyer, "*Standard Mathematical Tables*", CRC Press, 24th edition, pp. 398, 1976.

APPENDIX A. BENCHTOP CVA UNCERTAINTY ANALYSIS

This appendix formally shows the benchtop CVA measurement uncertainty analysis. In Chapter 2, a comparison between the CVA turbulent boundary layer mean velocity profile is compared with the pressure probe boundary layer mean velocity profile and discrepancies can be seen between the two data sets. It is desired to see if the discrepancy seen is due to the measurement or if there is a difference measuring a boundary layer profile with the benchtop CVA instead of a pressure probe.

In this analysis, a velocity that was deemed a good representation for the entire set of data was chosen for investigation. This resulted in the analysis being done for $u = 30\ m/s$ with a corresponding $OHR \cong 1.70$. The possible measurements that can lead to errors in the mean velocity data are the ambient temperature, the effective resolution of the DAQ, the electric noise detected by the DAQ, and the ambient pressure. The uncertainty for ambient temperature is 1 degree Celsius, for the effective resolution of the DAQ is 0.6mV, the electrical noise detected by the DAQ is 1mV_{rms}, and for the ambient pressure is 3 mm Hg. The uncertainty in the mean velocity data and fluctuating velocity data will be calculated using Equation (A.1) and Equation (A.2) respectively

$$\delta u = \sqrt{\sum_{i}^{n} [f(z_i + \delta z_i) - f(z_i)]^2} \tag{A.1}$$

$$\delta u' = \sqrt{\sum_{i}^{n} [f(z_i + \delta z_i) - f(z_i)]^2}\ . \tag{A.2}$$

From the measurement uncertainty analysis conducted for a mean velocity value of $u = 30\ m/s$, a contribution of $\delta u = \pm 1.73\ m/s$ can be expected for the uncertainty

of the ambient temperature, $\delta u = \pm 0.13\ m/s$ can be expected for the uncertainty of the resolution of the DAQ, $\delta u = \pm 0.21\ m/s$ can be expected for the electrical noise detected by the DAQ, and $\delta u = \pm 0.22\ m/s$ can be expected for the uncertainty of the ambient pressure. The total uncertainty for the mean velocity measurement at $u = 30\ m/s$ put together, is $\delta u = \pm 1.76\ m/s$. The uncertainty values for the ambient temperature and ambient pressure were calculated using the thermal/electric model and the uncertainty values for the effective resolution of the DAQ and the electrical noise were calculated using the calibration function.

From the measurement uncertainty analysis conducted for the fluctuating velocity at $u = 30\ m/s$, a contribution of $\delta u' = \pm 0.09\ m/s$ can be expected for the uncertainty of the ambient temperature, $\delta u' = \pm 0.01\ m/s$ can be expected for the uncertainty of the resolution of the DAQ, $\delta u' = \pm 0.22\ m/s$ can be expected for the electrical noise detected by the DAQ, and $\delta u' = \pm 0.01\ m/s$ can be expected for the uncertainty of the ambient pressure. The total uncertainty for the fluctuating velocity measurement at $u = 30\ m/s$ put together, is $\delta u' = \pm 0.24\ m/s$. Again, the uncertainty values for the ambient temperature and ambient pressure were calculated using the thermal/electric model and the uncertainty values for the effective resolution of the DAQ and the electrical noise were calculated using the calibration function.

Note that these calculations are only valid for the CVA Benchtop experiments shown in Chapter 2. For the BLDS – CVA, data seen in Chapter 3 and 4, higher e' values were seen in the data measured so higher values of measurement uncertainties is expected.

APPENDIX B. BENCHTOP CVA DAUGHTERBOARD INSTRUCTIONS

Required Equipment

CVA Test Box w/ auxiliary CVA card
4 Banana to Banana cables
MCX to MCX cable
MCX to female BNC cable
2 Digital Volt Meter (DVM)
Hot-wire probe support
Hot-wire probe

Nomenclature

V_w (V) = Constant voltage value across hot-wire probe
I_w (mA) = Current through wire
R_w (Ω) = Wire resistance at operating temperature (hot)
R_∞ (Ω) = Wire resistance at room temperature (cold)

Output Conversions

System Parameter		Calculation Using Output from Binding Post
$V_{w,in}$ (V)	=	(5 V/V) x $\boldsymbol{V_{w,in}}$
$V_{w,out}$ (V)	=	(5 V/V) x $\boldsymbol{V_{w,out}}$
I_w (mA)	=	(24 mA/V) x $\boldsymbol{I_w}$ (V)

Warnings

1. Know the operating limit of the hot-wire before using the CVA Test Box (does **NOT** have burnout protection).
2. Do **NOT** turn sensor switch on without a hot-wire probe attached to the auxiliary CVA card.

Operation Procedure

1. Ensure the ***Pwr.***, ***Sensor***, and ***Stop*** selector switches are flipped down.
2. Connect the hot-wire probe to the hot-wire probe support. Then connect the hot-wire probe support to the BNC end of the MCX to BNC cable. Then connect the MCX end of the MCX to BNC cable to the MCX to MCX cable. Then connect the remaining end of the MCX to MCX cable to the auxiliary CVA card.
3. Flip ***Pwr.*** switch up to the ***On*** position to turn CVA Test Box on. ***Pwr.*** LED should now be lit green and the display on the CVA Test Box should read 0.495

V. Check that pressing and holding the black $\boldsymbol{-V_{w,out}}$ button results in the display on the CVA Text Box to read 0 V and pressing and holding the black $\boldsymbol{-I_w}$ button results in the display on the CVA Test Box to read ~ 0 mA.

4. Set $V_{w,in}$ to desired value by performing the following procedure. To incrementally increase the voltage, flip the ***Stop*** switch up to ***Step*** and then press and release the red $\boldsymbol{V_w}$ button. This is the default setting. To decrease the voltage incrementally from the current voltage setting, hold the red $\boldsymbol{V_w}$ button down while flipping the ***Stop*** switch up to ***Step***. Releasing the red $\boldsymbol{V_w}$ button now decreases $V_{w,in}$. Furthermore, this puts the CVA test box to decrease voltage mode. To further decrease the current voltage setting, press and release the red $\boldsymbol{V_w}$ button. To switch from decrease voltage mode to increase voltage mode, flip the ***Stop*** switch down and back up to ***Step*** without holding the red $\boldsymbol{V_w}$ button down. Flip the ***Stop*** switch down to the off position when the desired $V_{w,in}$ is set.

   Notes: The CVA test box can only increase and decrease $V_{w,in}$ by increments of 0.016 V.

$$R_{w,calculated} = \frac{\boldsymbol{V_{w,out}}\ (\text{Volts})/5\ \left(\frac{Volts}{Volt}\right)}{\boldsymbol{I_w}\ (Volts)/41.6\ \left(\frac{Volts}{Amp}\right)}$$

$R_{w,actual}$ (Ω) = $R_{w,calc\square lated}$ (Ω) – [box + MCX to MCX cable res. + MCX to female BNC cable res.] (0.40 Ω) – probe support res. (Ω) – internal probe res. (Ω)

$$OHR = \frac{R_{w,actual}}{R_{\infty}}$$

   Do NOT exceed an OHR > 2.0.

5. Turn airflow on and adjust velocity to the first measurement point.
6. Flip ***Sensor*** switch up to the ***On*** position. ***Sensor*** LED should now be lit red. Check that the set $V_{w,in}$ is what is experimentally seen across the hot-wire probe, $V_{w,out}$. Verify this by pressing and holding the black $\boldsymbol{-V_{w,out}}$ button and observe if the value from the display on the CVA Test Box is the same as when the black $\boldsymbol{-V_{w,out}}$ button is not held down.

7. Connect one banana to banana cable end to the binding post of $\boldsymbol{V_{w,out}}$ and the other end to the positive input of one of the DVMs. Connect the second banana to banana cable to the binding post of ***Gnd*** and the other end to the negative input of the same DVM**.** Connect another banana to banana cable to the binding post of $\boldsymbol{I_w}$ and the other end to the positive input of the second DVM. Then connect the last banana to banana cable to the binding post of ***Gnd.*** and the other end to the negative input of the second DVM. Turn on the DVMs and record the mean of $\boldsymbol{V_{w,out}}$ and $\boldsymbol{I_w}$ using the min-max functions on the DVM. Repeat, as desired, for other air velocity values.
8. **Once measurements are completed, turn the *Sensor* and *Pwr.* switch down to the off position.**
9. Once **ALL** the LED lights are dark. The airflow can be turned off. Turn DVM off.
10. Carefully remove the hot-wire probe from the hot-wire probe support and return the hot-wire probe back to its original case. Disassemble all connections made with the cables and return them to their storage location.

National Aperture MM-3M-F Motor

MotionControl catalog **MotorizedStages**

MM-3M-F

Folded Motorized MicroMini™ Stages

Specifications:

	Standard Slider	AB Slider (anti-backlash)
*Repeatability:	±2µm	±0.5µm
*Homing Repeatability:	±2µm	±0.5µm
*Accuracy (linearity):	±3µm/inch	±1.5µm/inch
Speed (max.):	12mm/second @ 12V	1.65mm/second @ 12V
†Slider Backlash:	50µm	3µm
Encoder Conversion (resolution):	0.49609µm/count	0.12406µm/count
(See also: gearhead options)	with 16:1 gearhead	with 64:1 gearhead

*Encoder resolution must be added , based on the gearhead: 16:1 add ± 0.5µm, 64:1 add ±0.12 µm
†Slider backlash represents maximum overshoot

Runout (max.):	3µm/25.4mm
Gearhead Backlash:	1-2µm equivalent; can be compensated in software without overshoot
Motor:	10 mm diameter, 6-12 VDC servo, brush type (see motor specifications)
Vacuum compatiblity:	10^{-3} Torr, standard,10^{-6} Torr available
Load Capacity:	
Direct top or side load:	0.5 kg
Push:	0.5 kg
Pull:	0.5 kg
Tilt:	8 inch-ounce (560 gram-centimeter)
Twist:	4 inch-ounce (280 gram-centimeter)

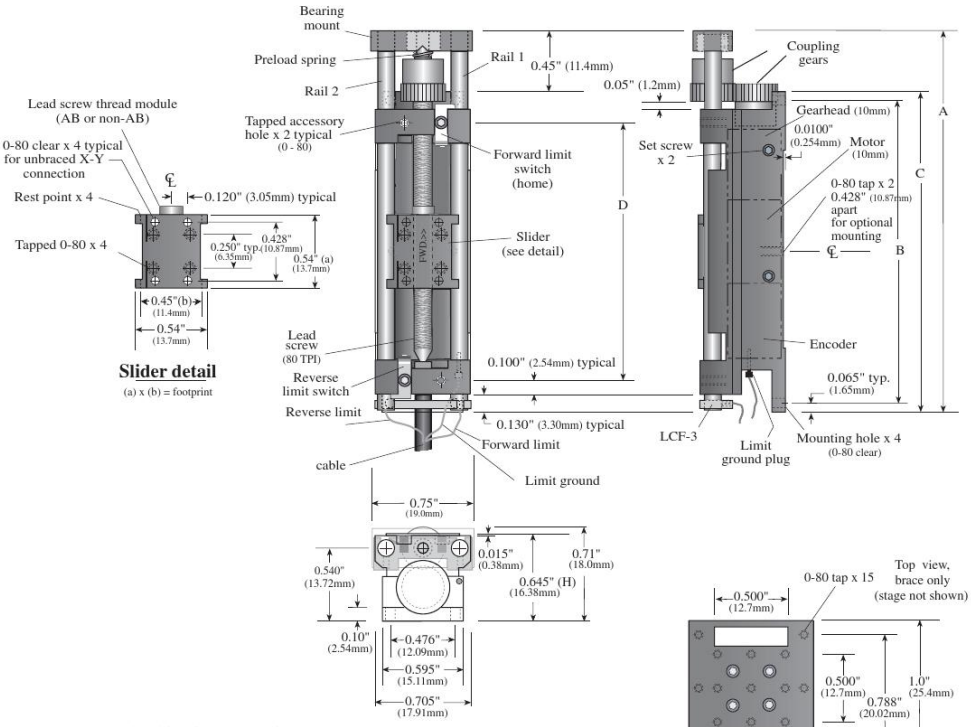
Note: These stages may be run at twice the specified ratings without damage, but with a loss of accuracy and speed.

Travel Ranges and Dimensions:

Model No.	Travel Range	Stage Body (L x W x H)	Weight
MM-3M-F-0.5	12.7mm (0.5 inch)	58.9 x 19.1 x 16.3 (mm)	53g
		2.32 x 0.75 x 0.64 (inch)	53g
MM-3M-F-1	25.4mm (1.0 inch)	71.6 x 19.1 x 16.3 (mm)	58g
		2.82 x 0.75 x 0.64 (inch)	58g
MM-3M-F-1.5	38.1mm (1.5 inch)	84.3 x 19.1 x 16.3 (mm)	63g
		3.32 x 0.75 x 0.64 (inch)	63g
MM-3M-F-2	50.8mm (2.0 inch)	97.0 x 19.1 x 16.3 (mm)	68g
		3.82 x 0.75 x 0.64 (inch)	68g
MM-3M-F-2.5	63.5mm (2.5 inch)	109.7 x 19.1 x 16.3 (mm)	73g
		4.32 x 0.75 x 0.64 (inch)	73g

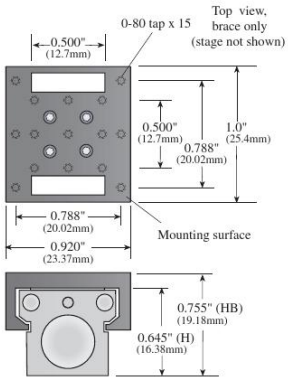
x, xy, xyz, xz configurations available
Specify -AB for Anti-Backlash

National Aperture, Inc. ▪ 16 Northwestern Dr. ▪ Salem, N.H. 03079-4810 ▪ Tel. (800) 360-4598 ▪ (603) 893-7393 ▪ FAX (603) 893-7857 ▪ **www.nationalaperture.com/www.naimotion.com**



Note: Symmetry is applicable wherever implied

	T (Travel)	A	B	C	D
MM-3M-F-0.5	0.5 inch	2.32	1.730	1.86	1.400
	12.7mm	58.9	43.94	47.2	35.56
MM-3M-F-1.0	1.0 inch	2.82	2.230	2.36	1.900
	25.4mm	71.6	56.64	59.9	48.26
MM-3M-F-1.5	1.5 inch	3.32	2.730	2.86	2.400
	38.1mm	84.3	69.34	72.6	60.96
MM-3M-F-2.0	2.0 inch	3.82	3.230	3.36	2.900
	50.8 mm	97.0	82.04	85.3	73.66
MM-3M-F-2.5	2.5 inch	4.32	3.730	3.86	3.400
	63.5 mm	109.7	94.74	98.0	86.36



BR-3M-X
Optional
Crash-resistant Slider Brace

Note:
Single axis brace is essential where attachments are vulnerable to crash.

All Sensor 10 INCH-G P4V-MINI Differential Sensors

MINIATURE AMPLIFIED LOW PRESSURE SENSORS

Low Pressure (1" H_2O to 30" H_2O) Sensors
PRIME GRADE

Features

- 0 to 1" H_2O to 0 to 30" H_2O Pressure Ranges
- Matched pressure port volumes
- Temperature Compensated (-25 to 85° C)
- Calibrated Zero and Span

Applications

- Medical Instrumentation
- Environmental Controls
- HVAC

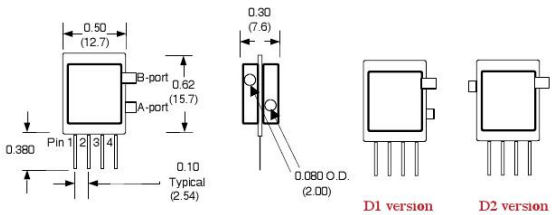
General Description

The Miniature Amplified Output pressure sensors is based upon a proprietary technology to reduce all output offset or common mode errors. This model provides a calibrated amplified output with superior output offset characteristics. Output offset errors due to change in temperature, stability to warm-up, stability to long time period, and position sensitivity are all significantly reduced when compared to conventional compensation methods. In addition the sensor utilizes a silicon, micromachined, stress concentration enhanced structure to provide a very linear output to measured pressure.

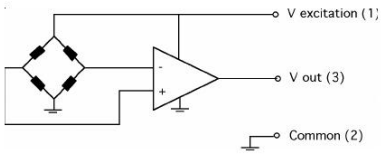
These calibrated and temperature compensated sensors give an accurate and stable output over a wide temperature range. This series is intended for use with non-corrosive, non-ionic working fluids such as air, dry gases and the like.

The output of the device is ratiometric to the supply voltage and operation from any D.C. supply voltage between 4.5 and 5.5 volts.

Physical Dimensions



Equivalent Circuit



ALL SENSORS
A 16035 Vineyard Blvd. Morgan Hill, CA 95037 P 408 225 4314 F 408 225 2079 E www.allsensors.com

Pressure Sensor Characteristics Maximum Ratings

Supply Supply Voltage VS	+4.5 to +5.5 Vdc
Common-mode pressure	10 psig
Lead Temperature (soldering 2-4 sec.)	250°C

Environmental Specifications

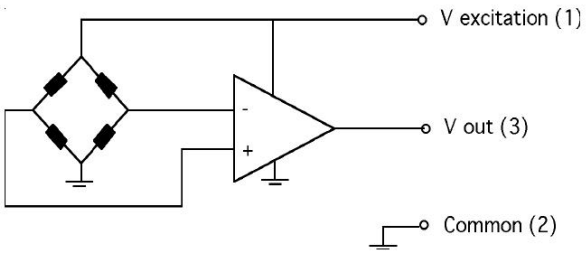
Temperature Ranges	
Compensated	-25 to 85° C
Operating	-40 to 125° C
Storage	-40 to 125° C
Humidity Limits	0 to 95% RH (non condensing)

Standard Pressure Ranges

Part Number	Operating Pressure	Nominal Span	Proof Pressure	Burst Pressure
1 INCH-Dx-P4V-MINI	±1 In H2O	4 V	100 In H2O	200 In H2O
1 INCH-G-P4V-MINI	0-1 In H2O	4 V	300 In H2O	200 In H2O
5 INCH-Dx-P4V-MINI	±5 In H2O	4 V	200 In H2O	300 In H2O
5 INCH-G-P4V-MINI	0-5 In H2O	4 V	200 In H2O	300 In H2O
10 INCH-Dx-P4V-MINI	±10 In H2O	4 V	200 In H2O	300 In H2O
10 INCH-G-P4V-MINI	0-10 In H2O	4 V	200 In H2O	300 In H2O
20 INCH-Dx-P4V-MINI	±20 In H2O	4 V	300 In H2O	500 In H2O
20 INCH-G-P4V-MINI	0-20 In H2O	4 V	300 In H2O	500 In H2O
30 INCH-Dx-P4V-MINI	±30 In H2O	4 V	500 In H2O	800 In H2O
30 INCH-G-P4V-MINI	0-30 In H2O	4 V	500 In H2O	800 In H2O
60 INCH-Dx-P4V-MINI	±60 In H2O	4V	500 In H2O	800 In H2O

For differential pressure D1 is the package with two pressure ports the same side, D2 has two ports the opposite sides.

Equivalent Circuit



Performance Characteristics for 1 INCH-Dx-P4V-MINI

Parameter, note 1	Minimum	Nominal	Maximum	Units
Operating Range, differential pressure		±1.0		"H2O
Output Span, note 5	±1.90	±2.0	±2.10	volt
Offset Voltage @ zero differential pressure	2.15	2.25	2.35	volt
Offset Temperature Shift (-25°C-85°C), note 2			±60	mvolt
Offset Warm-up Shift, note 3		±10		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±10		mvolt
Linearity, hysteresis error, note 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), note 2			±2	%span

Performance Characteristics for 1 INCH-G-P4V-MINI

Parameter, note 1	Minimum	Nominal	Maximum	Units
Operating Range, gage pressure		1.0		"H2O
Output Span, note 5	3.90	4.0	4.10	volt
Offset Voltage @ zero pressure	0.15	0.25	0.35	volt
Offset Temperature Shift (-25°C-85°C), note 2			±60	mvolt
Offset Warm-up Shift, note 3		±10		mvolt
Offset Position Sensitivity (±1g)		±15		mvolt
Offset Long Term Drift (one year)		±10		mvolt
Linearity, hysteresis error, note 4		0.05	0.25	%fs
Span Shift (5°C-50°C), note 2			±2	%span

Performance Characteristics for 5 INCH-Dx-P4V-MINI

Parameter, note 1	Minimum	Nominal	Maximum	Units
Operating Range, differential pressure		±5.0		"H2O
Output Span, note 5	±1.90	±2.0	±2.10	volt
Offset Voltage @ zero differential pressure	2.15	2.25	2.35	volt
Offset Temperature Shift (-25°C-85°C), note 2			±40	mvolt
Offset Warm-up Shift, note 3		±5		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±5		mvolt
Linearity, hysteresis error, note 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), note 2			±1	%span

ALL SENSORS
A 16035 Vineyard Blvd. Morgan Hill, CA 95037 P 408 225 4314 F 408 225 2079 E www.allsensors.com

Performance Characteristics for: 5 INCH-G-P4V-MINI

Parameter, NOTE 1	Minimum	Nominal	Maximum	Units
Operating Range, gage pressure		5.0		mbar
Output Span, NOTE 5	3.90	4.0	4.10	volt
Offset Voltage @ zero pressure	0.15	0.25	0.35	volt
Offset Temperature Shift (-25°C-85°C), NOTE 2			±40	mvolt
Offset Warm-up Shift, NOTE 3		±5		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±5		mvolt
Linearity, hysteresis error, NOTE 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), NOTE 2			±1	%span

Performance Characteristics for: 10 INCH-Dx-P4V-MINI

Parameter, NOTE 1	Minimum	Nominal	Maximum	Units
Operating Range, differential pressure		±10.0		mbar
Output Span, NOTE 5	±1.90	±2.0	±2.10	volt
Offset Voltage @ zero differential pressure	2.15	2.25	2.35	volt
Offset Temperature Shift (-25°C-85°C), NOTE 2			±20	mvolt
Offset Warm-up Shift, NOTE 3		±5		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±5		mvolt
Linearity, hysteresis error, NOTE 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), NOTE 2			±1	%span

Performance Characteristics for: 10 INCH-G-P4V-MINI

Parameter, NOTE 1	Minimum	Nominal	Maximum	Units
Operating Range, gage pressure		10.0		mbar
Output Span, NOTE 5	3.90	4.0	4.10	volt
Offset Voltage @ zero pressure	0.15	0.25	0.35	volt
Offset Temperature Shift (-25°C-85°C), NOTE 2			±20	mvolt
Offset Warm-up Shift, NOTE 3		±5		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±5		mvolt
Linearity, hysteresis error, NOTE 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), NOTE 2			±1	%span

Performance Characteristics for 20 INCH-Dx-P4V-MINI

Parameter, note 1	Minimum	Nominal	Maximum	Units
Operating Range, differential pressure		±20.0		"H2O
Output Span, note 5	±1.90	±2.0	±2.10	volt
Offset Voltage @ zero differential pressure	2.15	2.25	2.35	volt
Offset Temperature Shift (-25°C-85°C), note 2			±20	mvolt
Offset Warm-up Shift, note 3		±5		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±5		mvolt
Linearity, hysteresis error, note 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), note 2			±1	%span

Performance Characteristics for 20 INCH-G-P4V-MINI

Parameter, note 1	Minimum	Nominal	Maximum	Units
Operating Range, gage pressure		20.0		"H2O
Output Span, note 5	3.90	4.0	4.1	volt
Offset Voltage @ zero pressure	0.15	0.25	0.35	volt
Offset Temperature Shift (-25°C-85°C), note 2			±20	mvolt
Offset Warm-up Shift, note 3		±5		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±5		mvolt
Linearity, hysteresis error, note 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), note 2			±1	%span

Performance Characteristics for 30 INCH-Dx-P4V-MINI

Parameter, note 1	Minimum	Nominal	Maximum	Units
Operating Range, differential pressure		±30.0		"H2O
Output Span, note 5	±1.90	±2.0	±2.10	volt
Offset Voltage @ zero differential pressure	2.15	2.25	2.35	volt
Offset Temperature Shift (-25°C-85°C), note 2			±20	mvolt
Offset Warm-up Shift, note 3		±5		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±5		mvolt
Linearity, hysteresis error, note 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), note 2			±1	%span

A 16035 Vineyard Blvd. Morgan Hill, CA 95037 P 408 225 4314 F 408 225 2079 E www.allsensors.com

Performance Characteristics for 30 INCH-G-P4V-MINI

Parameter, NOTE 1	Minimum	Nominal	Maximum	Units
Operating Range, gage pressure		30.0		"H2O
Output Span, NOTE 5	3.9	4.0	4.1	volt
Offset Voltage @ zero pressure	0.15	0.25	0.35	volt
Offset Temperature Shift (-25°C-85°C), NOTE 2			±20	mvolt
Offset Warm-up Shift, NOTE 3		±5		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±5		mvolt
Linearity, hysteresis error, NOTE 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), NOTE 2			±1	%span

Performance Characteristics for 60 INCH-Dx-P4V-MINI

Parameter, note 1	Minimum	Nominal	Maximum	Units
Operating Range, differential pressure		±60.0		"H2O
Output Span, note 5	±1.90	±2.0	±2.10	volt
Offset Voltage @ zero differential pressure	2.15	2.25	2.35	volt
Offset Temperature Shift (-25°C-85°C), note 2			±20	mvolt
Offset Warm-up Shift, note 3		±5		mvolt
Offset Position Sensitivity (±1g)		±5		mvolt
Offset Long Term Drift (one year)		±5		mvolt
Linearity, hysteresis error, note 4		0.05	0.25	%fs
Span Shift (-25°C-85°C), note 2			±1	%span

Pressure Response: for any pressure applied the response time to get to 90% of pressure applied is typically less than 100 useconds.

Specification Notes

NOTE 1: ALL PARAMETERS ARE MEASURED AT 5.0 VOLT EXCITATION, FOR THE NOMINAL FULL SCALE PRESSURE AND ROOM TEMPERATURE UNLESS OTHERWISE SPECIFIED. PRESSURE MEASUREMENTS ARE WITH POSITIVE PRESSURE APPLIED TO THE FRONT PORT.

NOTE 2: SHIFT IS RELATIVE TO 25°C.

NOTE 3: SHIFT IS WITHIN THE FIRST HOUR OF EXCITATION APPLIED TO THE DEVICE.

NOTE 4: MEASURED AT ONE-HALF FULL SCALE RATED PRESSURE USING BEST STRAIGHT LINE CURVE FIT.

NOTE 5: THE VOLTAGE ADDED TO THE OFFSET VOLTAGE AT FULL SCALE PRESSURE. NOMINALLY THE OUTPUT VOLTAGE RANGE IS 0.25 TO 4.25 VOLTS FOR MINUS TO PLUS FULL SCALE PRESSURE.

Honeywell SSCDRNN015PAAA5 15 psia Absolute Sensors

Honeywell

TruStability® Silicon Pressure Sensors: SSC Series–Standard Accuracy

±2% Total Error Band, Amplified Compensated Digital Output, 1 psi to 150 psi (60 mbar to 10 bar)

DESCRIPTION

The TruStability® Standard Accuracy Silicon Ceramic (SSC) Series is a piezoresistive silicon pressure sensor offering a digital output for reading pressure over the specified full scale pressure span and temperature range.

The SSC Series is fully calibrated and temperature compensated for sensor offset, sensitivity, temperature effects, and non-linearity using an on-board Application Specific Integrated Circuit (ASIC). Calibrated output values for pressure are updated at approximately 2 kHz.

The SSC Series is calibrated over the temperature range of -20 °C to 85 °C [-4 °F to 185 °F]. The sensor is characterized for operation from a single power supply of either 3.3 Vdc or 5.0 Vdc.

These sensors measure absolute, differential, and gage pressures. The absolute versions have an internal vacuum reference and an output value proportional to absolute pressure. Differential versions allow application of pressure to either side of the sensing diaphragm. Gage versions are referenced to atmospheric pressure and provide an output proportional to pressure variations from atmosphere.

The TruStability® pressure sensors are intended for use with non-corrosive, non-ionic gases, such as air and other dry gases. An available option extends the performance of these sensors to non-corrosive, non-ionic liquids.

All products are designed and manufactured according to ISO 9001 standards.

FEATURES

- Industry-leading long-term stability
- Extremely tight accuracy of ±0.25% FSS BFSL (Full Scale Span Best Fit Straight Line)
- Total error band of ±2% full scale span maximum
- Modular and flexible design offers customers a variety of package styles and options, all with the same industry-leading performance specifications
- Miniature 10 mm x 10 mm [0.39 in x 0.39 in] package
- Low operating voltage
- Extremely low power consumption
- I^2C- or SPI- compatible 14-bit digital output (min. 12-bit sensor resolution)
- Precision ASIC conditioning and temperature compensated over -20 °C to 85 °C [-4 °F to 185 °F] temperature range
- RoHS compliant
- Virtually insensitive to mounting orientation
- Internal diagnostic functions increase system reliability
- Also available with analog output
- Absolute, differential, and gage types
- Pressure ranges from 1 psi to 150 psi (60 mbar to 10 bar)
- Custom calibration available
- Various pressure port options
- Liquid media option

POTENTIAL APPLICATIONS

- **Medical:**
  - Airflow monitors
  - Anesthesia machines
  - Blood analysis machines
  - Gas chromatography
  - Gas flow instrumentation
  - Kidney dialysis machines
  - Oxygen concentrators
  - Pneumatic controls
  - Respiratory machines
  - Sleep apnea equipment
  - Ventilators
- **Industrial:**
  - Barometry
  - Flow calibrators
  - Gas chromatography
  - Gas flow instrumentation
  - HVAC
  - Life sciences
  - Pneumatic controls

Table 1. Absolute Maximum Ratings[1]

Parameter	Min.	Max.	Unit
Supply voltage (V_{supply})	-0.3	6.0	Vdc
Voltage on any pin	-0.3	V_{supply} + 0.3	V
Digital interface clock frequency:			
I^2C	100	400	kHz
SPI	50	800	
ESD susceptibility (human body model)	3	-	kV
Storage temperature	-40 [-40]	85 [185]	°C [°F]
Soldering time and temperature:			
Lead solder (SIP, DIP)	4 s max. at 250 °C [482 °F]		
Peak reflow (SMT)	15 s max. at 250 °C [482 °F]		

Table 2. Operating Specifications

Parameter	Min.	Typ.	Max.	Unit
Supply voltage (V_{supply})[2]:				
3.3 Vdc	3.0	3.3[3]	3.6	Vdc
5.0 Vdc	4.75	5.0[3]	5.25	
Sensors are either 3.3 Vdc or 5.0 Vdc based on listing selected.				
Supply current:				
3.3 Vdc supply	-	1.6	2.1	mA
5.0 Vdc supply	-	2	3	
Compensated temperature range[4]	-20 [-4]	-	85 [185]	°C [°F]
Operating temperature range[5]	-40 [-40]	-	85 [185]	°C [°F]
Startup time (power up to data ready)	-	2.8	7.3	ms
Response time	-	0.46	-	ms
I^2C voltage level low	-	-	0.2	V_{supply}
I^2C voltage level high	0.8	-	-	V_{supply}
Pull up on SDA and SCL	1	-	-	kOhm
Accuracy[6]	-	-	±0.25	%FSS BFSL
Total error band[7]	-	-	±2	%FSS[8]
Output resolution	12	-	-	bits

Table 3. Environmental Specifications

Parameter	Characteristic
Humidity:	
Dry gases only (See "Options N and D" in Figure 1.)	0% to 95% RH, non-condensing
Liquid media (See "Options T and V" in Figure 1.)	100% condensing or direct liquid media on Port 1
Vibration	MIL-STD-202F, Curve AK (20.7 g random)
Shock	MIL-STD-202F, Method 213B, Condition F
Life[9]	1 million cycles minimum
Solder reflow	J-STD-020-C

Table 4. Wetted Materials[10]

Parameter	Port 1 (Pressure Port)	Port 2 (Reference Port)
Covers	high temperature polyamide	high temperature polyamide
Substrate	alumina ceramic	alumina ceramic
Adhesives	epoxy, silicone	epoxy, silicone
Electronic components	ceramic, glass, solder, silicon	silicon, glass, gold, solder

Notes:

1. Absolute maximum ratings are the extreme limits the device will withstand without damage.
2. Ratiometricity of the sensor (the ability of the digital device to maintain performance parameters independent of supply voltage) is achieved within the specified operating voltage for each option.
3. The sensor is not reverse polarity protected. Incorrect application of supply voltage or ground to the wrong pin may cause electrical failure.
4. The compensated temperature range is the temperature range over which the sensor will produce an output proportional to pressure within the specified performance limits.
5. The operating temperature range is the temperature range over which the sensor will produce an output proportional to pressure but may not remain within the specified performance limits.
6. Accuracy: The maximum deviation in output from a Best Fit Straight Line (BFSL) fitted to the output measured over the pressure range at 25 °C [77 °F]. Includes all errors due to pressure non-linearity, pressure hysteresis, and non-repeatability.
7. Total Error Band: The maximum deviation from the ideal transfer function over the entire compensated temperature and pressure range. Includes all errors due to offset, full scale span, pressure non-linearity, pressure hysteresis, repeatability, thermal effect on offset, thermal effect on span, and thermal hysteresis.
8. Full Scale Span (FSS) is the algebraic difference between the output signal measured at the maximum (Pmax.) and minimum (Pmin.) limits of the pressure range. (See Figure 1 for ranges.)
9. Life may vary depending on specific application in which sensor is utilized.
10. Contact Honeywell Customer Service for detailed material information.

CAUTION

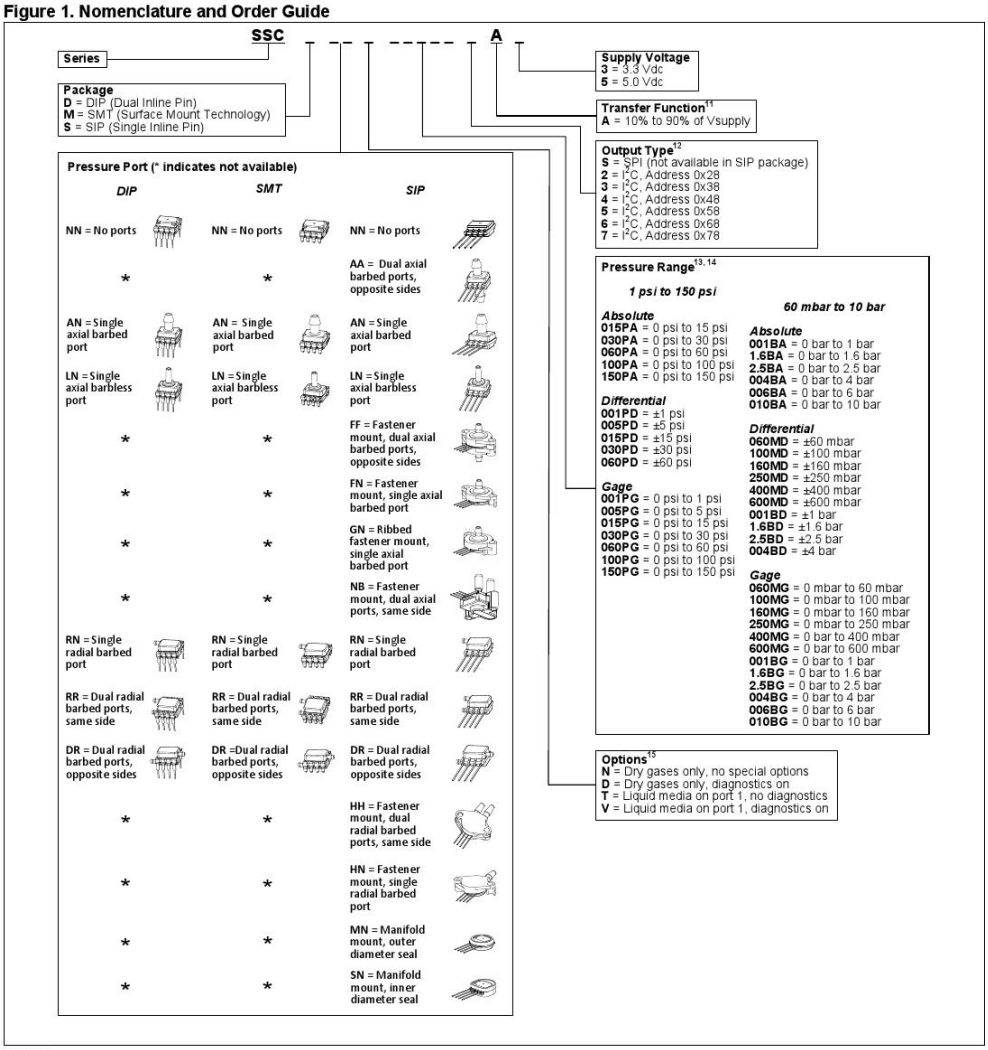
PRODUCT DAMAGE

- Ensure liquid media is applied to Port 1 only; Port 2 is not compatible with liquids.
- Ensure liquid media contains no particulates. All TruStability® sensors are dead-ended devices. Particulates can accumulate inside the sensor, causing damage or affecting sensor output.
- Recommend that the sensor be positioned with Port 1 facing downwards; any particulates in the system are less likely to enter and settle within the pressure sensor if it is in this position.
- Ensure liquid media does not create a residue when dried; build-up inside the sensor may affect sensor output. Rinsing of a dead-ended sensor is difficult and has limited effectiveness for removing residue.
- Ensure liquid media are compatible with wetted materials. Non-compatible liquid media will degrade sensor performance and may lead to sensor failure.

Failure to comply with these instructions may result in product damage.

TruStability® Silicon Pressure Sensors: SSC Series–Standard Accuracy

Figure 1. Nomenclature and Order Guide

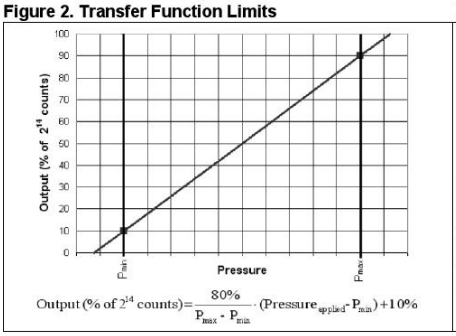


Notes:

11. The transfer function limits define the output of the sensor at a given pressure input. By specifying Pmin. and Pmax., the output at Pmin. and Pmax., the complete transfer function of the sensor is defined. See Figure 2 for a graphical representation of the transfer function. Other transfer functions are available. Contact Honeywell Customer Service for more information.
12. Analog output is also available. Contact Honeywell Customer Service for more information.
13. Custom pressure ranges are available. Contact Honeywell Customer Service for more information.
14. See Table 5 for an explanation of sensor pressure types.
15. See **CAUTION** on previous page.

±2% Total Error Band, Digital Output, 1 psi to 150 psi (60 mbar to 10 bar)

Figure 2. Transfer Function Limits



$$\text{Output (\% of } 2^{14} \text{ counts)} = \frac{80\%}{P_{max} - P_{min}} \cdot (\text{Pressure}_{applied} - P_{min}) + 10\%$$

Figure 3. Completed Catalog Listing Example

SSCSANN100PGSA3
SIP package, AN pressure port, no diagnostics, 100 psi gage, SPI output, 10% to 90% calibration at 3.3 Vdc.

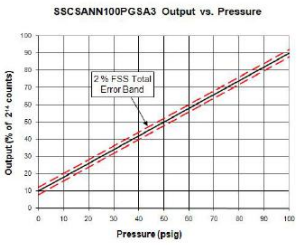


Table 5. Pressure Types

Pressure Type	Description
Absolute	Output is proportional to the difference between applied pressure and a built-in reference to vacuum. $P_{min.}$ is set at absolute zero pressure (full vacuum).
Differential	Output is proportional to the difference between the pressures applied to each port. (Port 1 – Port 2) 50% point of transfer function set at Port 1 = Port 2.
Gage	Output is proportional to the difference between applied pressure and atmospheric (ambient) pressure. $P_{min.}$ is set at atmospheric pressure.

Table 6. Sensor Output at Significant Percentages

% Output	Digital Counts (decimal)	Digital Counts (hex)
0	0	0x0000
10	1638	0x0666
50	8192	0x2000
90	14746	0x399A
100	16383	0x3FFF

Table 7. Pressure Range Specifications for 1 psi to 150 psi

Order Code	Pressure Range		Over-pressure[16]	Burst Pressure[17]	Common Mode Pressure[18]	Long-term Stability (1000 hr, 25 °C [77 °F])
	$P_{min.}$	$P_{max.}$				
Absolute						
015PA	0 psi	15 psi	30 psi	60 psi	NA	±0.25% FSS
030PA	0 psi	30 psi	60 psi	120 psi	NA	±0.25% FSS
060PA	0 psi	60 psi	120 psi	240 psi	NA	±0.25% FSS
100PA	0 psi	100 psi	250 psi	250 psi	NA	±0.25% FSS
150PA	0 psi	150 psi	250 psi	250 psi	NA	±0.25% FSS
Differential						
001PD	-1 psi	1 psi	10 psi	10 psi	150 psi	±0.35% FSS
005PD	-5 psi	5 psi	30 psi	40 psi	150 psi	±0.35% FSS
015PD	-15 psi	15 psi	30 psi	60 psi	150 psi	±0.25% FSS
030PD	-30 psi	30 psi	60 psi	120 psi	150 psi	±0.25% FSS
060PD	-60 psi	60 psi	120 psi	240 psi	250 psi	±0.25% FSS
Gage						
001PG	0 psi	1 psi	10 psi	10 psi	150 psi	±0.35% FSS
005PG	0 psi	5 psi	30 psi	40 psi	150 psi	±0.35% FSS
015PG	0 psi	15 psi	30 psi	60 psi	150 psi	±0.25% FSS
030PG	0 psi	30 psi	60 psi	120 psi	150 psi	±0.25% FSS
060PG	0 psi	60 psi	120 psi	240 psi	250 psi	±0.25% FSS
100PG	0 psi	100 psi	250 psi	250 psi	250 psi	±0.25% FSS
150PG	0 psi	150 psi	250 psi	250 psi	250 psi	±0.25% FSS

Table 8. Pressure Range Specifications for 60 mbar to 10 bar

Order Code	Pressure Range		Over-pressure[16]	Burst Pressure[17]	Common Mode Pressure[18]	Long-term Stability (1000 hr, 25 °C [77 °F])
	$P_{min.}$	$P_{max.}$				
Absolute						
001BA	0 bar	1 bar	2 bar	4 bar	NA	±0.25% FSS
1.6BA	0 bar	1.6 bar	4 bar	8 bar	NA	±0.25% FSS
2.5BA	0 bar	2.5 bar	6 bar	8 bar	NA	±0.25% FSS
004BA	0 bar	4 bar	8 bar	16 bar	NA	±0.25% FSS
006BA	0 bar	6 bar	17 bar	17 bar	NA	±0.25% FSS
010BA	0 bar	10 bar	17 bar	17 bar	NA	±0.25% FSS
Differential						
060MD	-60 mbar	60 mbar	500 mbar	700 mbar	10 bar	±0.35% FSS
100MD	-100 mbar	100 mbar	500 mbar	700 mbar	10 bar	±0.35% FSS
160MD	-160 mbar	160 mbar	500 mbar	700 mbar	10 bar	±0.35% FSS
250MD	-250 mbar	250 mbar	1.4 bar	2.5 bar	10 bar	±0.35% FSS
400MD	-400 mbar	400 mbar	1.4 bar	2.5 bar	10 bar	±0.35% FSS
600MD	-600 mbar	600 mbar	2 bar	4 bar	10 bar	±0.25% FSS
001BD	-1 bar	1 bar	2 bar	4 bar	10 bar	±0.25% FSS
1.6BD	-1.6 bar	1.6 bar	4 bar	8 bar	10 bar	±0.25% FSS
2.5BD	-2.5 bar	2.5 bar	6 bar	8 bar	10 bar	±0.25% FSS
004BD	-4 bar	4 bar	8 bar	16 bar	10 bar	±0.25% FSS
Gage						
060MG	0 mbar	60 mbar	500 mbar	700 mbar	3.5 bar	±0.35% FSS
100MG	0 mbar	100 mbar	500 mbar	700 mbar	10 bar	±0.35% FSS
160MG	0 mbar	160 mbar	500 mbar	700 mbar	10 bar	±0.35% FSS
250MG	0 mbar	250 mbar	1.4 bar	2.5 bar	10 bar	±0.35% FSS
400MG	0 mbar	400 mbar	1.4 bar	2.5 bar	10 bar	±0.35% FSS
600MG	0 mbar	600 mbar	2 bar	4 bar	10 bar	±0.35% FSS
001BG	0 bar	1 bar	2 bar	4 bar	10 bar	±0.25% FSS
1.6BG	0 bar	1.6 bar	4 bar	8 bar	10 bar	±0.25% FSS
2.5BG	0 bar	2.5 bar	6 bar	8 bar	10 bar	±0.25% FSS
004BG	0 bar	4 bar	8 bar	16 bar	16 bar	±0.25% FSS
006BG	0 bar	6 bar	17 bar	17 bar	17 bar	±0.25% FSS
010BG	0 bar	10 bar	17 bar	17 bar	17 bar	±0.25% FSS

Notes:

16. Overpressure: The maximum pressure which may safely be applied to the product for it to remain in specification once pressure is returned to the operating pressure range. Exposure to higher pressures may cause permanent damage to the product. Unless otherwise specified this applies to all available pressure ports at any temperature with the operating temperature range.
17. Burst pressure: The maximum pressure that may be applied to any port of the product without causing escape of pressure media. Product should not be expected to function after exposure to any pressure beyond the burst pressure.
18. Common mode pressure: The maximum pressure that can be applied simultaneously to both ports of a differential pressure sensor without causing changes in specified performance.

Table 9. Pinout for SMT and DIP Packages

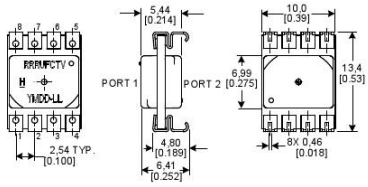
Output Type	Pin 1	Pin 2	Pin 3	Pin 4	Pin 5	Pin 6	Pin 7	Pin 8
I^2C	GND	V_{supply}	SDA	SCL	NC	NC	NC	NC
SPI	GND	V_{supply}	MISO	SCLK	SS	NC	NC	NC

Table 10. Pinout for SIP Package

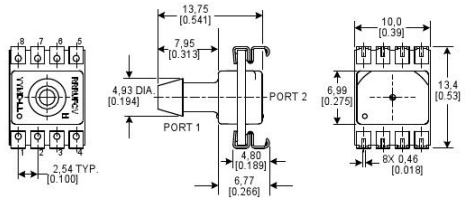
Output Type	Pin 1	Pin 2	Pin 3	Pin 4
I^2C	GND	V_{supply}	SDA	SCL

Figure 5. SMT Pressure Port Dimensional Drawings (For reference only: mm [in])

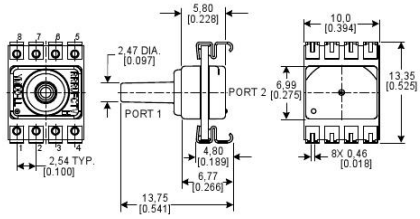
SMT NN: No ports



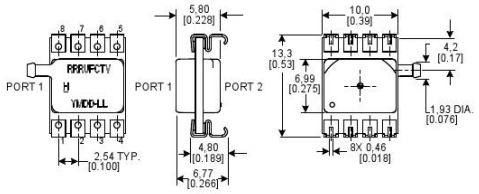
SMT AN: Single axial barbed port



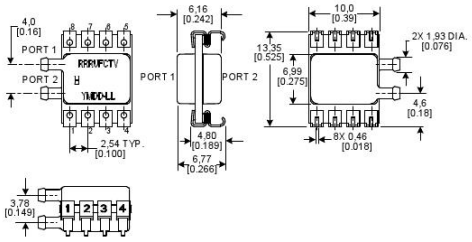
SMT LN: Single axial barbless port



SMT RN: Single radial barbed port



SMT RR: Dual radial barbed ports, same side



SMT DR: Dual radial barbed ports, opposite sides

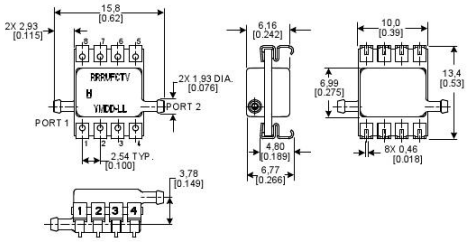
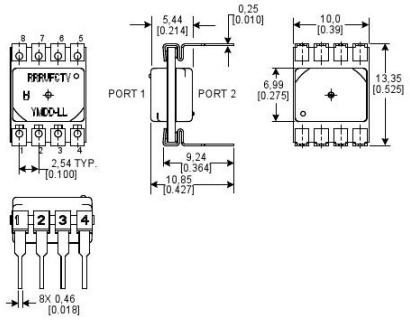
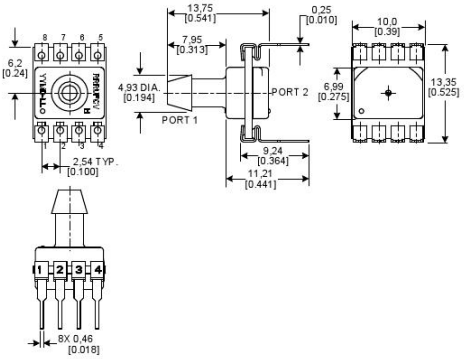


Figure 4. DIP Pressure Port Dimensional Drawings (For reference only: mm [in])

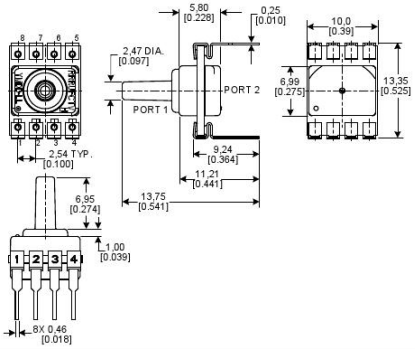
DIP NN: No ports



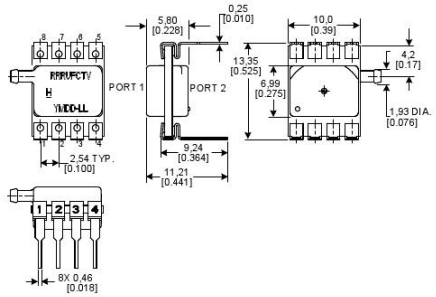
DIP AN: Single axial barbed port



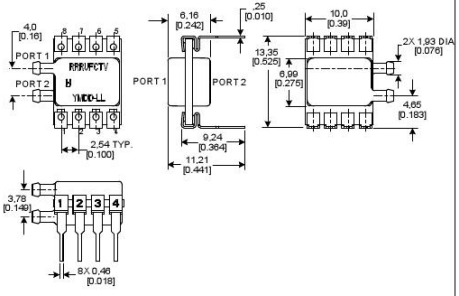
DIP LN: Single axial barbless port



DIP RN: Single radial barbed port



DIP RR: Dual radial barbed ports, same side



DIP DR: Dual radial barbed ports, opposite sides

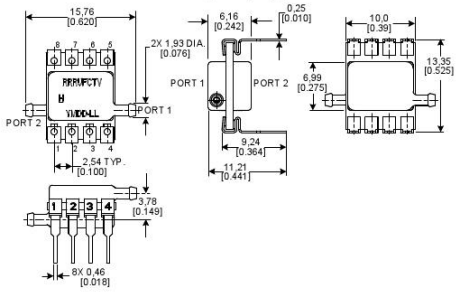
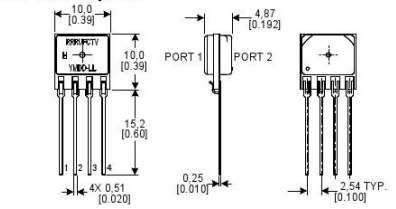
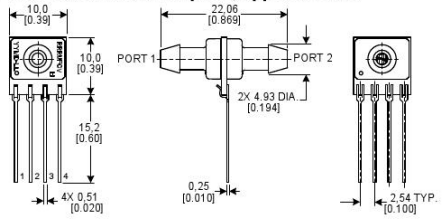


Figure 6. SIP Package Dimensional Drawings (For reference only: mm [in])

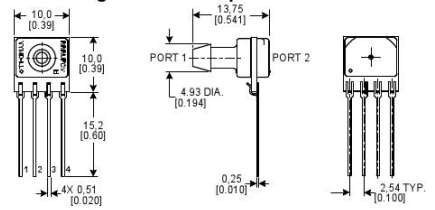
SIP NN: No ports



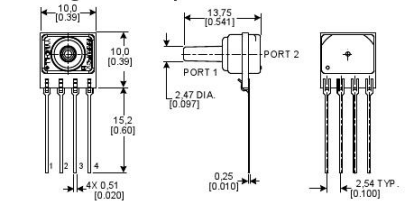
SIP AA: Dual axial barbed ports, opposite sides



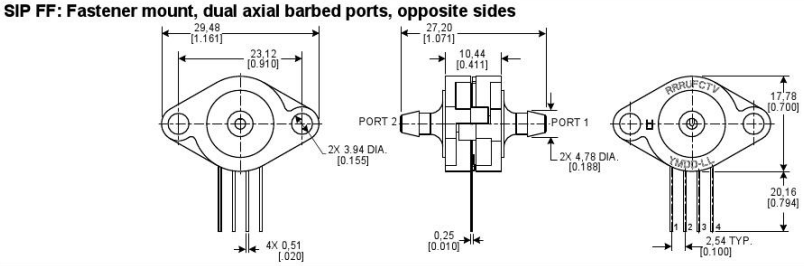
SIP AN: Single axial barbed port



SIP LN: Single barbless port



SIP FF: Fastener mount, dual axial barbed ports, opposite sides



SIP FN: Fastener mount, single axial barbed port

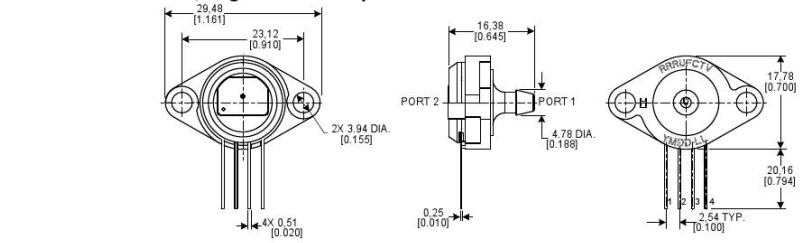
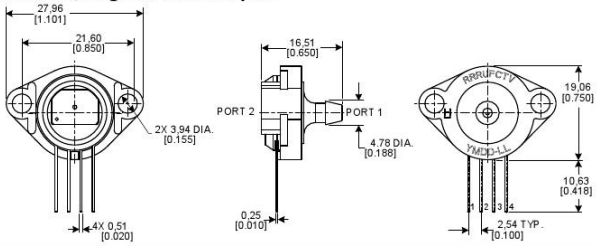
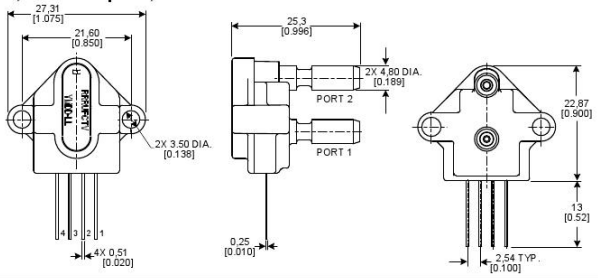


Figure 6. SIP Package Dimensional Drawings (continued)

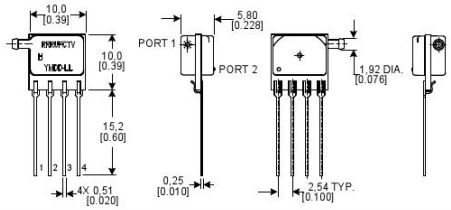
SIP GN: Ribbed fastener mount, single axial barbed port



SIP NB: Fastener mount, dual axial ports, same side



SIP RN: Single radial barbed port



SIP RR: Dual radial barbed ports, same side

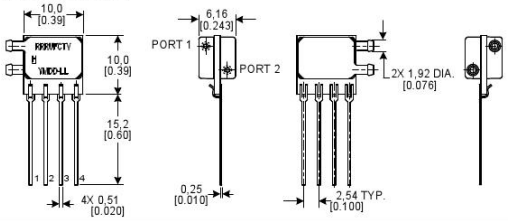
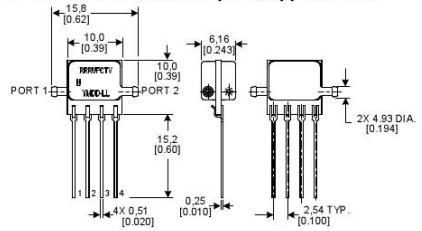
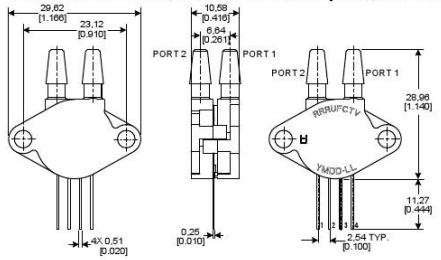


Figure 6. SIP Package Dimensional Drawings (continued)

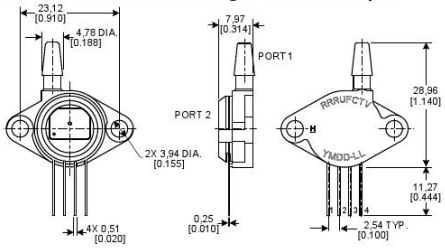
SIP DR: Dual radial barbed ports, opposite side



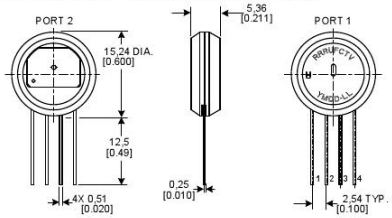
SIP HH: Fastener mount, dual radial barbed ports, same side



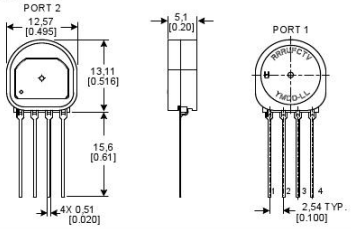
SIP HN: Fastener mount, single radial barbed port



SIP MN: Manifold mount, outer diameter seal



SIP SN: Manifold mount, inner diameter seal



⚠ WARNING

PERSONAL INJURY

DO NOT USE these products as safety or emergency stop devices or in any other application where failure of the product could result in personal injury.

Failure to comply with these instructions could result in death or serious injury.

WARRANTY/REMEDY

Honeywell warrants goods of its manufacture as being free of defective materials and faulty workmanship. Honeywell's standard product warranty applies unless agreed to otherwise by Honeywell in writing; please refer to your order acknowledgement or consult your local sales office for specific warranty details. If warranted goods are returned to Honeywell during the period of coverage, Honeywell will repair or replace, at its option, without charge those items it finds defective. **The foregoing is buyer's sole remedy and is in lieu of all other warranties, expressed or implied, including those of merchantability and fitness for a particular purpose. In no event shall Honeywell be liable for consequential, special, or indirect damages.**

While we provide application assistance personally, through our literature and the Honeywell web site, it is up to the customer to determine the suitability of the product in the application.

Specifications may change without notice. The information we supply is believed to be accurate and reliable as of this printing. However, we assume no responsibility for its use.

⚠ WARNING

MISUSE OF DOCUMENTATION

- The information presented in this product sheet is for reference only. DO NOT USE this document as a product installation guide.
- Complete installation, operation, and maintenance information is provided in the instructions supplied with each product.

Failure to comply with these instructions could result in death or serious injury.

SALES AND SERVICE

Honeywell serves its customers through a worldwide network of sales offices, representatives and distributors. For application assistance, current specifications, pricing or name of the nearest Authorized Distributor, contact your local sales office or:

E-mail: info.sc@honeywell.com

Internet: www.honeywell.com/sensing

Phone and Fax:

Asia Pacific	+65 6355-2828; +65 6445-3033 Fax
Europe	+44 (0) 1698 481481; +44 (0) 1698 481676 Fax
Latin America	+1-305-805-8188; +1-305-883-8257 Fax
USA/Canada	+1-800-537-6945; +1-815-235-6847 +1-815-235-6545 Fax

Sensing and Control
Honeywell
1985 Douglas Drive North
Golden Valley, MN 55422
www.honeywell.com

008213-2-EN IL50 GLO Printed in USA
March 2011

Honeywell

Energizer 9-Volt $LiMnO_2$ battery

PRODUCT DATASHEET

Energizer

1-800-383-7323 USA/CAN
www.energizer.com

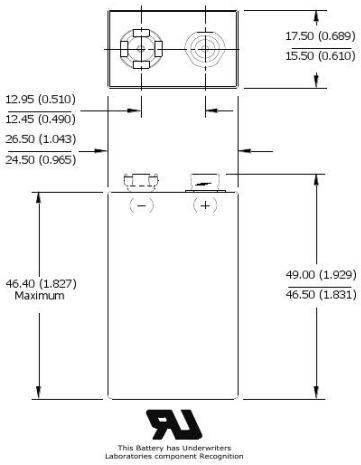
ENERGIZER LA522

Advanced Lithium

9V

Industry Standard Dimensions

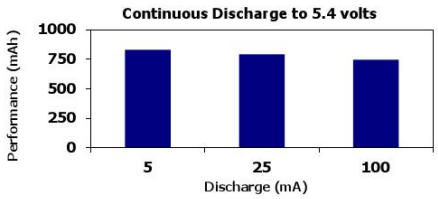
mm (inches)

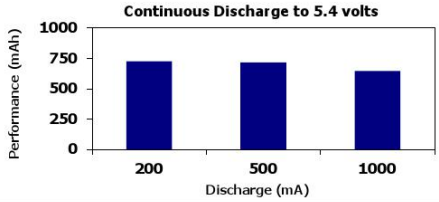


Specifications

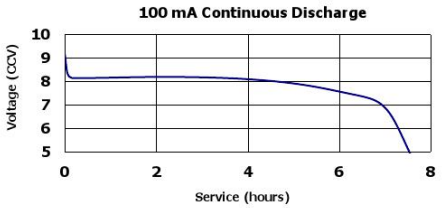
Classification:	Lithium 9V
Chemical System:	Lithium-Manganese Dioxide (Li/MnO_2)
Designation:	ANSI-1604LC
Nominal Voltage:	9.0 volts
Operating Temp:	-40°C to 60°C (-40°F to 140°F)
Storage Temp:	-40°C to 60°C (-40°F to 140°F)
Max Discharge :	1000 mA continuous
Safety Features:	Positive Temperature Coefficient Switch (PTC) Burst Proof Venting Holes
Typical Weight:	33.9 grams (1.2 oz.)
Typical Volume:	21.4 cubic centimeters (1.3 cubic inch)
Jacket:	Plastic Label
Terminal:	Miniature Snap
Shelf Life:	10 Years
Typical Li Content:	1.35 grams (0.048 oz.)
UL/UN Listed:	MH47482

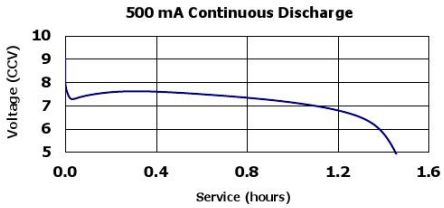
Milliamp-Hours Performance (21°C)





Typical Discharge Performance (21°C)





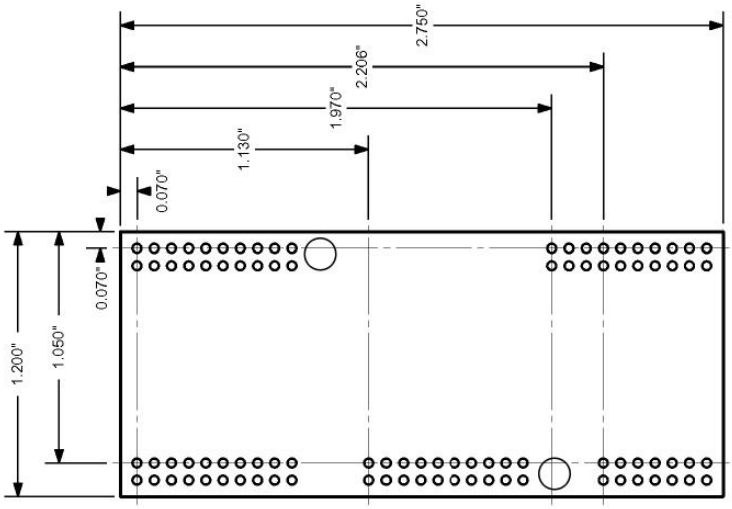
Important Notice

This datasheet contains typical information specific to products manufactured at the time of its publication.

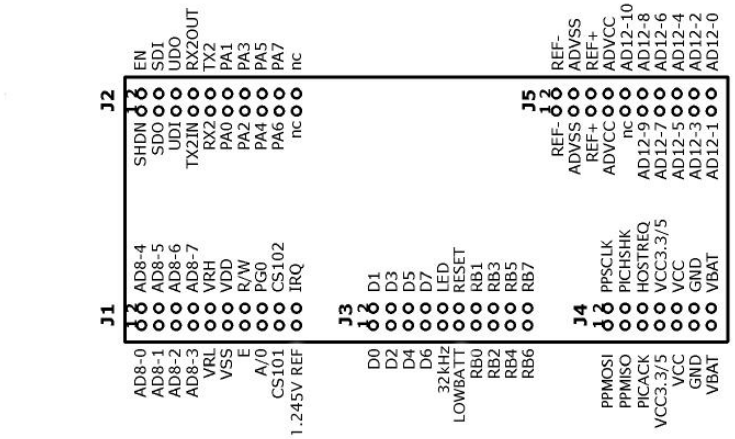
Form No. EBC - 4217

Page 1 of 1

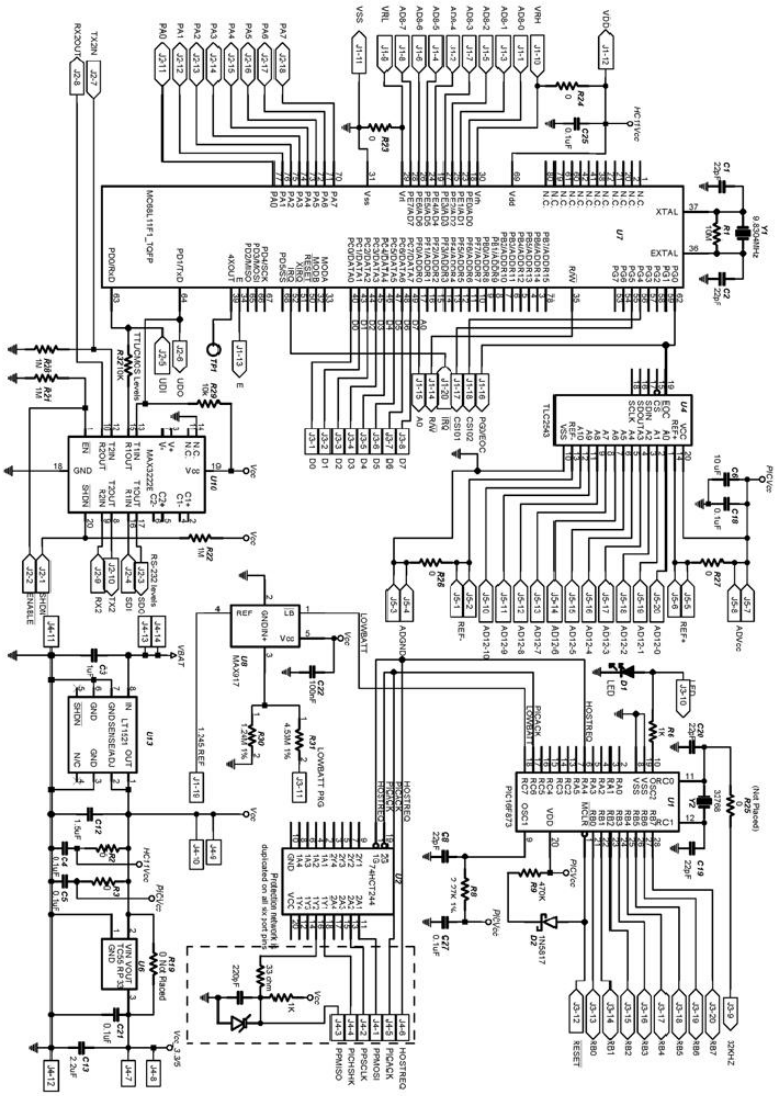
TFX-11v2 Board Pinout and Dimensions



TOP VIEW



TFX-11v2 Schematic



APPENDIX D. BLDS – CVA SOFTWARE

BLDS – GT – CVA Hardware Test Program

```
//BLDS-GT-CVA Hardware Test Program
//Updated 6_28_2013 added CVA daughterboard functions
//
////////////// DEFINITIONS FOR  -GT-CVA (November 2012/ Jan 2013) HARDWARE CONFIGURATION //////////////
// ... PA & RB digital inputs/outputs (PA0-7 map to 0-7 and RB0-7 map to 16 to 23)
// 0 PA0 i counter
// 1 PA1 o CVA--Vw step
// 2 PA2 i stage upper limit
// 3 PA3 o CVA--Vw stop/set
// 4 PA4 i stage lower limit
// 5 PA5 o CVA-- sensor ON
// 6 PA6 i rotary home if ON
// 7 PA7 o CVA--power ON
// 16 RB0 o stage power
// 17 RB1 o turn on 3 pressure sensors, LED, bat voltage divider, temp sensor
// 18 RB2 o encoder power... MUST be ON to count encoder pulses!
// 19 RB3 o stage direction
// 20 RB4 o rotary step
// 21 RB5 o satellite R power (also used for Conrad power)
// 22 RB6 o satellite L power
// 23 RB7 o rotary power
// ... AD12 12-bit A/D inputs (read with CHAN(**), **=0-10 (11 channels))
// 0 AD12-0 temperature
// 1 AD12-1 R satellite probe differential, also CVA-- Vw in check
// 2 AD12-2 R satellite freestream differential,  also CVA-- Vw out
// 3 AD12-3 battery voltage
// 4 AD12-4 R satellite static, also CVA-- Iw out
// 5 AD12-5 main static
// 6 AD12-6 L satellite static
// 7 AD12-7  freestream differential
// 8 AD12-8 L satellite freestream differential
// 9 AD12-9 main probe differential
// 10 AD12-10 L satellite probe differential
////////////// -AV3b (through BLDS-G Dec 2009) HARDWARE CONFIGURATION //////////////
// ... PA & RB digital inputs/outputs (PA0-7 map to 0-7 and RB0-7 map to 16 to 23
// 0 PA0 i counter
// 1 PA1 o turn on 3 pressure sensors, LED, bat voltage divider, temp sensor
// 2 PA2 o stage power
// 3 PA3 o satellite R power (also used for Conrad power)
// 4 PA4 o stage direction AND rotary step
// 5 PA5 i rotary home if ON
```

```
// 6 PA6 i stage lower limit
// 7 PA7 i stage upper limit
// 16 RB0
// 17 RB1 o satellite L power
// 18 RB2
// 19 RB3 o rotary power... also, rotary home encoder power
// 20 RB4
// 21 RB5 o encoder power... MUST be ON to count encoder pulses!
// 22 RB6
// 23 RB7 o stage extra power (smaller resistor for current limit)
// ... AD12 12-bit A/D inputs (read with CHAN(**), **=0-10 (11 channels))
// 0 AD12-0 main static
// 1 AD12-1 R satellite probe differential
// 2 AD12-2 R satellite freestream differential
// 3 AD12-3 battery voltage
// 4 AD12-4 R satellite static
// 5 AD12-5 main freestream differential
// 6 AD12-6 L satellite static
// 7 AD12-7 temperature
// 8 AD12-8 L satellite freestream differential
// 9 AD12-9 main probe differential
// 10 AD12-10 L satellite probe differential
        print "***BLDS-GT CVA Hardware Test Program: AV3 JUN 2013 ***"
        gosub startup   // Output power off, Check Data File Erased,
                // Initialize Control-C Redirection & Onerr Redirection
        gosub mainmenu
        gosub shutdown          // does not return
mainmenu:
        pclr 1, 3, 5, 7, 16, 17, 18, 19, 20, 21, 22, 23             // Make sure no power is
applied to any outputs
        choice1$ = "N"
        while (choice1$ <> "5") // loop main menu until user exits
                gosub printmainmenu
                input "Menu Choice: " choice1$, #1
                if (choice1$ = "1")
                        gosub sensorsmenu
                else
                        if (choice1$ = "2")
                                gosub hotwiremenu
                        else
                                if (choice1$ = "3")
                                        gosub stagemenu
                                else
                                        if (choice1$ = "4")
                                                gosub rotarymenu
                                        else
```

```
                                        if (choice1$ = "5")
                                                print "Exiting..."
                                        else
                                                print nl$, "Invalid choice!  Please re-
enter."
                                        endif
                                endif
                        endif
                endif
        wend
        // while loop terminated, Menu choice "5" was entered by user
return

printmainmenu:
        print nl$, "Main Menu"
        print " 1) Pressure Sensors/Battery"
        print " 2) Hot-Wire"
        print " 3) Stage Assembly"
        print " 4) Rotary Assembly"
        print " 5) Exit Program"
return

sensorsmenu:
        choice2$ = "N"
        while (choice2$ <> "5") // loop sensors menu until user exits
                gosub printsensorsmenu
                input "Menu Choice: " choice2$, #1
                if (choice2$ = "1")
                        gosub sensorsoption1
                else
                        if (choice2$ = "2")
                                gosub sensorsoption2
                        else
                                if (choice2$ = "3")
                                        gosub sensorsoption3
                                else
                                        if (choice2$ = "4")
                                                gosub sensorsoption4
                                        else
                                                if (choice2$ = "5")
                                                        print "Exiting..."
                                                else
                                                        print nl$, "Invalid choice!  Please re-
enter."
                                                endif
```

```
                                        endif
                                    endif
                                endif
                            endif
        wend
return

printsensorsmenu:
        print nl$, "Sensors Menu"
        print " 1) Read all sensors at max rate for (N) seconds, average and display
results"
        print " 2) Read selected sensor, display all raw readings"
        print " 3) Read selected sensor, display ave / min-max / std dev"
        print " 4) Flash LED"
        print " 5) Exit Sensors Menu"
return

sensorsoption1:
        print nl$, "Read all sensors at max rate for (N) seconds, average and display
results"
        //turn ON power to main sensors and slight warm-up delay from gathering user
input
        pset 17
        sleep 0
        sleep 20
        // from user get number of seconds to test sensors
        answer$ = "N"
        while (answer$ <> "Y")
                input "Number of seconds to read sensors: " duration
                print "You entered ", duration
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        // initialize sensor variables
        pitotsignal! = 0
        pitotave! = 0
        staticsignal! = 0
        staticave! = 0
        prestonsignal! = 0
        prestonave! = 0
        Te! = 0
        tempave! = 0
        batv! = 0
        batvave! = 0
        starttime = ?           // Start Max Wait Timer
        endtime = ?                     // endtime used to calculate elapsed time
```

```
        k = 1                           // number of iterations so far (used in running ave
calculation)
        while ( endtime - starttime < duration )
                // read & average main sensors
                kk! = k
                pitotsignal! = chan(7)
                pitotave! = pitotave! + ((pitotsignal! - pitotave!) / kk!)
                staticsignal! = chan(5)
                staticave! = staticave! + ((staticsignal! - staticave!) / kk!)
                prestonsignal! = chan(9)
                prestonave! = prestonave! + ((prestonsignal! - prestonave!) / kk!)
                Te! = chan(0)
                tempave! = tempave! + ((Te! - tempave!) / kk!)
                batv! = chan(3)
                batvave! = batvave! + ((batv! - batvave!) / kk!)
                k = k + 1               // increment our iteration counter
                endtime = ?     // capture the current time
        wend
        // turn off power to main sensors
        pclr 17
        // compute sensor avg voltages, battery avg voltages, & temp in C
        pitot! = (pitotave! * .0000763)
        static! = (staticave! * .0000763)
        preston! = (prestonave! * .0000763)
        T! = (tempave! * .00763) - 273.0            // 100.0 * (tempave! * 0.0000763) -
273.0
        batvf! = (batvave! * 3 * .0000763)          // 1/3 divider circuit in -AV3
        // print data to screen
        pitots$ = str(#7.3F, pitot!, ",")
        statics$ = str(#7.3F, static!, ",")
        prestons$ = str(#7.3F, preston!, ",")
        temps$ = str(#7.2F, T!, ",")
        batvfs$ = str(#7.3F, batvf!)
        print "Pitot(V),Static(V),Preston(V),Temp(C),Voltage"
        print pitots$, statics$, prestons$, temps$, batvfs$
return
sensorsoption2:
        print nl$, "Read selected sensor, display all raw readings"
        //turn ON power to main sensors with slight warm-up delay from gathering user
input
        pset 17
        sleep 0
        sleep 20
        // from user get which sensor to test
        answer$ = "N"
        while (answer$ <> "Y")
```

```
                print "Please choose which sensor to test"
                print "Choices: pitot(7), static(5), preston(9), temperature(0), battery(3)"
                input "Please choose which sensor to test: " sensor_to_test, #1
                print "You entered ", sensor_to_test
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        print
        for k = 1 to 50
                print chan(sensor_to_test)
                sleep 0
                sleep 1
        next k
        print
        // turn off power to main sensors
        pclr 17
return

sensorsoption3:
        print nl$, "Read selected sensor, display ave / min-max / std dev"
        //turn ON power to main sensors with slight warm-up delay from gathering user
input
        pset 17
        sleep 0
        sleep 20
        // from user get which sensor to test
        answer$ = "N"
        while (answer$ <> "Y")
                print "Please choose which sensor to test"
                print "Choices: pitot(7), static(5), preston(9), temperature(0), battery(3)"
                input "Please choose which sensor to test: " sensor_to_test, #1
                print "You entered ", sensor_to_test
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        sensorave! = 0
        sdevave! = 0
        sumOfSquaresAve! = 0
        sensormin! = chan(sensor_to_test) // get initial reading
        sensormax! = sensormin!
        for k = 1 to 50
                sleep 0
                sleep 1
                sensorsignal! = chan(sensor_to_test)
                if (sensorsignal! > sensormax!)
                        sensormax! = sensorsignal!
```

```
                endif
                if (sensorsignal! < sensormin!)
                        sensormin! = sensorsignal!
                endif
                kk! = k
                sensorave! = sensorave! + ((sensorsignal! - sensorave!) / kk!)
                sumOfSquaresAve! = sumOfSquaresAve! + ((sensorsignal! *
sensorsignal! - sumOfSquaresAve!) / kk!)
        next k
        // turn off power to main sensors
        pclr 17
        sdevave! = sqr(sumOfSquaresAve! - (sensorave! * sensorave!))
        if (sensor_to_test = 3) // battery
                sdev! =   (sdevave!   * 3 * .0000763)           // 1/3 divider circuit in -AV3
                sensor! = (sensorave! * 3 * .0000763)
                min! =    (sensormin! * 3 * .0000763)
                max! =    (sensormax! * 3 * .0000763)
        else
                if (sensor_to_test = 0) // temperature
                        sdev! =   (sdevave!   * .00763)          // 100.0 * (tempave! *
0.0000763) - 273.0
                        sensor! = (sensorave! * .00763) - 273.0
                        min! =    (sensormin! * .00763) - 273.0
                        max! =    (sensormax! * .00763) - 273.0
                else // must be either pitot, preston, or static
                        sdev! =   (sdevave!   * .0000763)
                        sensor! = (sensorave! * .0000763)
                        min! =    (sensormin! * .0000763)
                        max! =    (sensormax! * .0000763)
                endif
        endif
        // print data to screen
        print
        print "ave  = ", str(#7.3F, sensor!)
        print "sdev = ", str(#7.3F, sdev!)
        print "min  = ", str(#7.3F, min!)
        print "max  = ", str(#7.2F, max!)
        print
return

sensorsoption4:
// 17 RB1 o turn on 3 pressure sensors, LED, bat voltage divider, temp sensor
        print nl$, "Flashing LED for 10 seconds..."
        for x = 1 to 10
                pset 17
                sleep 0
```

```
                sleep 50
                pclr 17
                sleep 0
                sleep 50
        next x
return

hotwiremenu:
        choice3$ = "N"
        while (choice3$ <> "6") // loop menu until user exits
                gosub printhotwiremenu
                input "Menu Choice: " choice3$, #1
                if (choice3$ = "1")
                        gosub hotwireoption1
                else
                        if (choice3$ = "2")
                                gosub hotwireoption2
                        else
                                if (choice3$ = "3")
                                        gosub hotwireoption3
                                else
                                        if (choice3$ = "4")
                                                gosub hotwireoption4
                                        else
                                                if (choice3$ = "5")
                                                        gosub hotwireoption5
                                                else
                                                        if (choice3$ = "6")
                                                                print "Exiting..."
                                                        else
                                                                print nl$, "Invalid choice!
Please re-enter."
                                                        endif
                                                endif
                                        endif
                                endif
                        endif
                endif
        wend
        pclr 1, 3, 5, 7
return

printhotwiremenu:
        print nl$, "Hot-Wire Menu"
        print " 1) Set/Check V_w"
        print " 2) Read all 3 CVA Outputs 5-sec avg (Have Hot-Wire Attached!!!)"
```

```
        print " 3) Read only selected sensor; display all raw readings (Have Hot-Wire
Attached!!!)"
        print " 4) Read selected signal, display average/min-max/std dev (Have Hot-Wire
Attached!!!)"
        print " 5) Calculate a suggested V_w based on ambient/testing conditions"
        print " 6) Exit Hot-Wire Menu"
return

hotwireoption1:
        pset 7 // Power PA7 CVA--power ON
        sleep 0
        sleep 20
        gosub setvw
        pclr 1, 3, 5, 7
return

hotwireoption2:
        pset 7 // Power PA7 CVA--power ON
        sleep 0
        sleep 20
        gosub setvw
        CVAvwout = 2
        CVAiwout = 4
        CVAvwin = 1
        input "Attach Hot-Wire to BLDS-GT-CVA if it hasn't been done already. Press
Enter to continue." blank
        pset 5
        sleep 0
        sleep 20
        // initialize sensor variables
        vwoutsignal! = 0
        vwoutave! = 0
        iwoutsignal! = 0
        iwoutave! = 0
        vwinsignal! = 0
        vwinave! = 0
        duration = 5    // seconds
        starttime = ?   // Start Max Wait Timer
        endtime = ?     // endtime used to calculate elapsed time
        k = 1           // number of iterations so far (used in running ave calculation)
        while ( endtime - starttime < duration )
                // read & average main sensors
                kk! = k
                vwoutsignal! = chan(CVAvwout)
                vwoutave! = vwoutave! + ((vwoutsignal! - vwoutave!) / kk!)
                iwoutsignal! = chan(CVAiwout)
```

```
                iwoutave! = iwoutave! + ((iwoutsignal! - iwoutave!) / kk!)
                vwinsignal! = chan(CVAvwin)
                vwinave! = vwinave! + ((vwinsignal! - vwinave!) / kk!)
                k = k + 1          // increment our iteration counter
                endtime = ?      // capture the current time
        wend
        pclr 1, 3, 5, 7
        // compute sensor avg voltages, battery avg voltages, & temp in C
        vwout! = (vwoutave! * .0000763 / 5)
        iwout! = (iwoutave! * .0000763)
        vwin! = (vwinave! * .0000763 / 5)
        // print data to screen
        vwouts$ = str(#7.3F, vwout!)
        iwouts$ = str(#7.3F, iwout!)
        vwins$ = str(#7.3F, vwin!)
        print
        print "V_wout   = ", vwouts$
        print "I_wout  = ", iwouts$
        print "V_win = ", vwins$
return
hotwireoption3:
        pset 7 // Power PA7 CVA--power ON
        sleep 0
        sleep 20
        gosub setvw
        // determine which sensor to use
        // sensor to use -> sensor_to_test
        gosub get_sensor
        input "Attach Hot-Wire to BLDS-GT-CVA if it hasn't been done already. Press
Enter to continue." blank
        pset 5
        sleep 0
        sleep 20
        print
        for k = 1 to 50
                print chan(sensor_to_test)
                sleep 0
                sleep 1
        next k
        print
        pclr 1, 3, 5, 7
return

hotwireoption4:
        pset 7 // Power PA7 CVA--power ON
        sleep 0
```

```
        sleep 20
        gosub setvw
        // determine which satellite and which sensor to use
        // satellite choic -> pwrpin
        // sensor to use   -> sensor_to_test
        gosub get_sensor
        input "Attach Hot-Wire to BLDS-GT-CVA if it hasn't been done already. Press
Enter to continue." blank
        pset 5
        sleep 0
        sleep 20
        sensorave! = 0
        sdevave! = 0
        sumOfSquaresAve! = 0
        sensormin! = chan(sensor_to_test) // get initial reading
        sensormax! = sensormin!
        for k = 1 to 50
                sleep 0
                sleep 1
                sensorsignal! = chan(sensor_to_test)
                if (sensorsignal! > sensormax!)
                        sensormax! = sensorsignal!
                endif
                if (sensorsignal! < sensormin!)
                        sensormin! = sensorsignal!
                endif
                kk! = k
                sensorave! = sensorave! + ((sensorsignal! - sensorave!) / kk!)
                sumOfSquaresAve! = sumOfSquaresAve! + ((sensorsignal! *
sensorsignal! - sumOfSquaresAve!) / k)
        next k
        pclr 1, 3, 5, 7
        sdevave! = sqr(sumOfSquaresAve! - (sensorave! * sensorave!))
        if (sensor_to_test = 2 | sensor_to_test = 1)
                sdev! =   (sdevave!   * .0000763 / 5)
                sensor! = (sensorave! * .0000763 / 5)
                min! =    (sensormin! * .0000763 / 5)
                max! =    (sensormax! * .0000763 / 5)
        else
                sdev! =   (sdevave!   * .0000763)
                sensor! = (sensorave! * .0000763)
                min! =    (sensormin! * .0000763)
                max! =    (sensormax! * .0000763)
        endif
        // print data to screen
        print
```

```
        print "ave  = ", str(#7.3F, sensor!)
        print "sdev = ", str(#7.3F, sdev!)
        print "min  = ", str(#7.3F, min!)
        print "max  = ", str(#7.2F, max!)
        print
return
hotwireoption5:
        pset 17
        Te! = 0
        input "What is the cold resistance (in ohms) of the hot-wire at 20 degree Celsius?
" R_inf!
        input "What is the maximum speed (in m/s) the hot-wire will experience during
the gathering of boundary layer profile data? " U!
        input "What is the ambient pressure (in mm Hg)? " P_inf!
        Ohr! = 2
        alpha_20! = 0.0042
        dia! = 0.0000038                        // meters
        L! = 0.00127                            // meters
        U! = U! * 0.1
        Te! = chan(0)
        T_inf! = (Te! * .00763)                 // Kelvin
        pclr 17
        R_w! = Ohr! * R_inf!
        T_w! = T_inf! + (((R_w!/R_inf!) - 1)/alpha_20!)
        T_f! = (T_w! + T_inf!)/2
        k_f! = 0.0239 + 0.000073189 * (T_f!-273.0) - 0.000000018982 * (T_f!-
273.0)*(T_f!-273.0)
        T_ratio! = T_f!/T_inf!
        rho_f! = 1.2929 * (273.0/T_f!) * (P_inf!/760)
        nu_f! = (84.986+(7 * T_f!) - 0.0037501 * (T_f! * T_f!)) * 0.00000001/rho_f!
        Re! = U!*dia!/nu_f!
        // Approximation of Re^0.45
        approx! = 1 + 0.45*log(Re!) + ((0.45*log(Re!)) * (0.45*log(Re!)))/2 +
((0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)))/6 + ((0.45*log(Re!)) *
(0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)))/24 + ((0.45*log(Re!)) *
(0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)))/120 +
((0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!))
* (0.45*log(Re!)))/720
        // Approximation of (T_f/T_inf)^0.17
        T_rat_approx! = 1 + 0.17*log(T_ratio!) + ((0.17*log(T_ratio!)) *
(0.17*log(T_ratio!)))/2 + ((0.17*log(T_ratio!)) * (0.17*log(T_ratio!)) *
(0.17*log(T_ratio!)))/6
        A_s! = 3.14159 * dia! * L!
        V_w! = sqr((0.24 + 0.56 * approx!) * T_rat_approx! * k_f! * A_s! * ((Ohr!-
1)/alpha_20!) * Ohr! * R_inf!/dia!)
        V_ws$ = str(#7.3F, V_w!, nl$)
```

```
        print nl$, "The suggested wire voltage to set the hot-wire to is: ", V_ws$
return
get_sensor:
        answer$ = "N"
        while (answer$ <> "Y")
                print "Which sensor would you like to test?"
                input "V_wout(U), I_wout(I), V_win (W): " probe$, #1
                print "You entered ", probe$
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        if (probe$ = "U")
                sensor_to_test = 2 // CVA-V_wout
        else
                if (probe$ = "I")
                        sensor_to_test = 4 // CVA-I_wout
                else
                        sensor_to_test = 1 // CVA-V_win
                endif
        endif
return
setvw:
        vwoutsignal! = 0
        answer$ = "N"
        while (answer$ <> "Y")
                input "What V_w would you like to test at?: " vwireset!
                print "You entered ", vwireset!
                input "Is this correct? <Y or N>: " answer$, #1
        wend

        vwinsignal! = chan(1)
        vwinsignal! = (vwinsignal! * .0000763 / 5)
        vwiresets! = vwireset! - 0.016
        if (vwiresets! >= vwinsignal!)
                while(vwiresets! > vwinsignal!)
                        pset 3
                        pset 1
                        pclr 1
                        pclr 3
                        vwinsignal! = chan(1)
                        vwinsignal! = (vwinsignal! * .0000763 / 5)
                wend
        else
                if (vwireset! < vwinsignal!)
                        pset 1
                        pset 3
```

```
                        pclr 1
                        vwinsignal! = chan(1)
                        vwinsignal! = (vwinsignal! * .0000763 / 5)
                        while(vwireset! <= vwinsignal!)
                                pset 1
                                pclr 1
                                vwinsignal! = chan(1)
                                vwinsignal! = (vwinsignal! * .0000763 / 5)
                        wend
                endif
        endif
        print "V_w is set to ", vwinsignal!
return
stagemenu:
        choice2$ = "N"
        while (choice2$ <> "5") // loop menu until user exits
                gosub printstagemenu
                input "Menu Choice: " choice2$, #1
                if (choice2$ = "1")
                        gosub stageoption1
                else
                        if (choice2$ = "2")
                                gosub stageoption2
                        else
                                if (choice2$ = "3")
                                        gosub stageoption3
                                else
                                        if (choice2$ = "4")
                                                gosub stageoption4
                                        else
                                                if (choice2$ = "5")
                                                        print "Exiting..."
                                                else
                                                        print nl$, "Invalid choice!  Please re-
enter."
                                                endif
                                        endif
                                endif
                        endif
                endif
        wend
        pclr 16 // ensure stage pwr off upon exit of stage menu
return

printstagemenu:
        // Display actual counts during all tests
```

```
        print nl$, "Stage Menu"
        print " 1) Move in selected direction for specified number of counts at set PWM"
        print " 2) Move to limit in selected direction at set PWM"
        print " 3) Move for specified time interval"
        print " 4) Move probe traverse"
        print " 5) Exit Stage Menu"
return

stageoption1:
        print nl$, "Move in selected direction for specified number of counts at set PWM"
        // prompt for direction
        answer$ = "N"
        while (answer$ <> "Y")
                print "Select the stage movement direction"
                print " 1) Up"
                print " 2) Down"
                input "Direction Choice: " stage_direction
                print "You entered ", stage_direction
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        if (stage_direction = 1)
                pclr 19, 20                // set direction UP
        else
                pset 19, 20      // set direction DOWN
        endif
        // prompt for target counts
        answer$ = "N"
        while (answer$ <> "Y")
                print "Input the target number of counts to move stage"
                input "Target counts: " target
                print "You entered ", target
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        // prompt for PWM
        answer$ = "N"
        while (answer$ <> "Y")
                print "Input the PWM Duty Cycle as an integer percentage"
                print "where 1 = 10% and 10 = 100%"
                input "Duty Cycle (1 - 10): " on_time
                print "You entered ", on_time
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        off_time = 10 - on_time
```

```
        pset 18 //turn ON stage encoder power
        sleep 0
        sleep 20
        // turn on background counting process to count encoder pulses
        backcount = 0
        count backcount
        while (backcount < target)                  // PWM loop
                pset 16                             // stage pwr on
                for i = 1 to on_time
                next i
                pclr 16                             // stage pwr off
                for i = 1 to off_time
                next i
        wend
        sleep 0                                     // give stage time to come to a stop
        sleep 50
        count                                       // turn off background counting
        pclr 18                                     // turn OFF encoder power to limit
battery power drain
        print "counts moved = ", backcount
return

stageoption2:
        print nl$, "Move to limit in selected direction at set PWM"
        // prompt for direction
        answer$ = "N"
        while (answer$ <> "Y")
                print "Select the stage movement direction"
                print " 1) Up"
                print " 2) Down"
                input "Direction Choice: " stage_direction, #1
                print "You entered ", stage_direction
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        if (stage_direction = 1)
                pclr 19, 20      // set direction UP
                checkpin = 2
        else
                pset 19, 20  // set direction DOWN
                checkpin = 4
        endif
        // prompt for PWM
        answer$ = "N"
        while (answer$ <> "Y")
                print "Input the PWM Duty Cycle as an integer percentage"
```

```
                print "where 1 = 10% and 10 = 100%"
                input "Duty Cycle (1 - 10): " on_time
                print "You entered ", on_time
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        off_time = 10 - on_time
        pset 18 //turn ON stage encoder power
        sleep 0
        sleep 20
        // turn on background counting process to count encoder pulses
        backcount = 0
        count backcount
        while (pin(checkpin) <> 0)  // finish PWM loop when stage limit pin = 0
                pset 16 // stage pwr on
                for i = 1 to on_time
                next i
                pclr 16 // stage pwr off
                for i = 1 to off_time
                next i
        wend
        sleep 0 // give stage time to come to a stop
        sleep 50
        count   // turn off background counting
        pclr 18 // turn OFF encoder power to limit battery power drain
        print "counts moved = ", backcount
return

stageoption3:
        print nl$, "Move for specified time interval"
        // prompt for direction
        answer$ = "N"
        while (answer$ <> "Y")
                print "Select the stage movement direction"
                print " 1) Up"
                print " 2) Down"
                input "Direction Choice: " stage_direction, #1
                print "You entered ", stage_direction
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        // prompt for stage movement duration in seconds
        answer$ = "N"
        while (answer$ <> "Y")
                print "Input the number of seconds to move stage"
                input "Duration in seconds: " duration, #1
```

```
                print "You entered ", duration
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        if (stage_direction = 1)
                pclr 19, 20                             // set direction UP
                checkpin = 2
        else
                pset 19, 20                     // set direction DOWN
                checkpin = 4
        endif
        pset 16 // stage pwr on
        readrtc // copy PIC time to ?-variable
        starttime = ?
        done = 0
        while (pin(checkpin) <> 0 & done <> 1)
                endtime = ?
                if ((endtime - starttime) > duration)
                        done = 1
                endif
        wend
        pclr 16                                         // stage pwr off
        sleep 0                                         // give stage time to come to a stop
        sleep 50
return

stageoption4:
        movement$ = "s"
        input "Enter the number of counts to move: " increment
        //increment = 100                       //set how many counts to move per input
        print "Press e to exit, i to move up, or k to move down"
        print "WARNING: Limit switches are not activated"
        while (movement$ <> "e" & movement$ <> "E")
                input "" movement$, #1, !N;
                // turn on background counting process to count encoder pulses
                pset 18                                 //turn on encoder
                backcount = 0
                count backcount
                if(movement$ = "I" | movement$ = "i")
                                        //reset count
                        pclr 19, 20     //set stage direction
                        pset 16         //turn on stage
                        while(backcount < increment)
                        wend
                        pclr 16
                        pclr 18
```

```
                else
                        if(movement$ = "K" | movement$ = "k")
                                pset 19, 20      //set stage direction
                                pset 16          //turn on stage
                                while(backcount < increment)
                                wend
                                pclr 16
                                pclr 18
                        endif
                endif
        wend
        pclr 19, 20
        pset 19, 20
return

rotarymenu:
        //RB7 enables step motor and provides hold torque (PA3 was used for this before
-AV3)
        pset 23 // RB7
        choice2$ = "N"
        while (choice2$ <> "4") // loop menu until user exits
                gosub printrotarymenu
                input "Menu Choice: " choice2$, #1
                if (choice2$ = "1")
                        gosub rotaryoption1
                else
                        if (choice2$ = "2")
                                gosub rotaryoption2
                        else
                                if (choice2$ = "3")
                                        gosub rotaryoption3
                                else
                                        if (choice2$ = "4")
                                                print "Exiting..."
                                        else
                                                print nl$, "Invalid choice!  Please re-enter."
                                        endif
                                endif
                        endif
                endif
        wend
        // Clear RB7 to turn off stepper motor--no hold torque now!
        pclr 23
return

printrotarymenu:
```

```
        print nl$, "Rotary Menu"
        print " 1) roll specified number of steps, HOME status at each step"
        print " 2) read/report HOME status"
        print " 3) roll to HOME, report number of steps"
        print " 4) Exit Rotary Menu"
return

rotaryoption1:
        print nl$, "roll specified number of steps, HOME status at each step"
        // prompt for stage movement duration in seconds
        answer$ = "N"
        while (answer$ <> "Y")
                print "Input the number of steps to roll the rotary assembly"
                input "Steps to roll: " steps
                print "You entered ", steps
                input "Input Y to continue, any other command will allow you to redo
input: " answer$, #1
        wend
        // RB3 & RB4 hi/lo steps the roll motor
        pclr 19, 20
        sleep 0
        sleep 20
        for counter = 1 to steps
                pset 19, 20
                sleep 0
                sleep 5
                pclr 19, 20
                sleep 0
                sleep 5
                // rotary home indicator is PA6; check with rotary motor enabled by
setting RB7
                homep = pin(6)
                if (homep = 0) // not a home position
                        print "Step ", counter, ": Not at HOME position"
                else
                        print "Step ", counter, ": At HOME position"
                endif
        next counter
        pclr 19, 20
        sleep 0
        sleep 20
return

rotaryoption2:
        print nl$, "read/report HOME status"
        // PA4 hi/lo steps the roll motor
```

```
        pclr 19, 20
        sleep 0
        sleep 20
        // rotary home indicator is PA6; check with rotary motor enabled by setting RB7
        homep = pin(6)
        if (homep = 0) // not a home position
                print "Not at HOME position"
        else
                print "At HOME position"
        endif
        pclr 19, 20
        sleep 0
        sleep 20
return

rotaryoption3:
        print nl$, "roll to HOME, report number of steps"
        // RB3 & RB7 hi/lo steps the roll motor
        pclr 19, 20
        sleep 0
        sleep 20
        counter = 0
        // rotary home indicator is PA6; check with rotary motor enabled by setting RB7
        homep = pin(6)
        while (homep = 0)
                pset 19, 20
                sleep 0
                sleep 5
                pclr 19, 20
                sleep 0
                sleep 5
                counter = counter + 1
                homep = pin(6)
        wend
        print counter, " steps taken to get to home position"
        pclr 19, 20
        sleep 0
        sleep 20
return

////////////////// STARTUP SUBROUTINE ////////////////////
startup:
        pclr 1, 3, 5, 7, 16, 17, 18, 19, 20, 21, 22, 23
        true = 1              // some true/false constants
        false = 0             // do not change these!!!
        cbreak quick_exit             // goto mainmenu label when Ctrl-C pressed
```

```
        error = 0
        onerr quick_exit, error         // goto quick_exit if execution error occurs,
                                                // error information stored in 'error'
        lowbatvalue! = 5.3              // Initialize low-battery threshhold
        nl$ = str(\13,\10)              // a new-line string for formatting output
        if (DFERASED = 0)
                print nl$, "WARNING: data file not erased"
                print "Enter Y to continue, otherwise program will terminate"
                input answer$,#1
                if (answer$ <> "Y")
                        stop
                endif
        endif
return
////////////////////////////////////////////////////
/////////////// QUICK_EXIT SUBROUTINE ////////////////////
// Control-C was pressed, OR error occured, exit gracefully
// Error List on page 144 of TFBasic Manual
quick_exit:
        pclr 1, 3, 5, 7, 16, 17, 18, 19, 20, 21, 22, 23             // Make sure no power is
applied to any outputs
        count                   // Make sure background counter is off
        call &hFD88,0           // flush output buffer to datafile
        if (error = 0)          // error did not occur
                print "Control-C Pressed, Program Stopped, Ready for Data Offload."
        else                            // error did occur
                message$ = str("Error #", error / 65536, "  @", #05H, error % 65536)
                print message$
                store #$, message$
                call &hFD88,0           // flush output buffer to datafile
        endif
        stop
return          // Program will never reach this return statement
////////////////////////////////////////////////////
/////////////// SHUTDOWN SUBROUTINE ////////////////////
shutdown:
        //CLEAR OUTPUTS, just to be sure
        pclr 1, 3, 5, 7, 16, 17, 18, 19, 20, 21, 22, 23
        stop    // go ahead and stop, power-on but in lo-power config
return
////////////////////////////////////////////////////
////////////////////////////
```

BLDS – GT – CVA Program

```
// BLDS-GT-CVA NOV 2012/ JAN 2103
////////////// DEFINITIONS FOR  -GT-CVA (November 2012/ Jan 2013) HARDWARE CONFIGURATION //////////////
// ... PA & RB digital inputs/outputs (PA0-7 map to 0-7 and RB0-7 map to 16 to 23)
// 0 PA0 i counter
// 1 PA1 o CVA--Vw step
// 2 PA2 i stage upper limit
// 3 PA3 o CVA--Vw stop/set
// 4 PA4 i stage lower limit
// 5 PA5 o CVA-- sensor ON
// 6 PA6 i rotary home if ON
// 7 PA7 o CVA--power ON
// 16 RB0 o stage power
// 17 RB1 o turn on 3 pressure sensors, LED, bat voltage divider, temp sensor
// 18 RB2 o encoder power... MUST be ON to count encoder pulses!
// 19 RB3 o stage direction
// 20 RB4 o rotary step
// 21 RB5 o satellite R power (also used for Conrad power)
// 22 RB6 o satellite L power
// 23 RB7 o rotary power
// ... AD12 12-bit A/D inputs (read with CHAN(**), **=0-10 (11 channels)
// 0 AD12-0 temperature
// 1 AD12-1 R satellite probe differential, also CVA-- Vw in check
// 2 AD12-2 R satellite freestream differential,  also CVA-- Vw out
// 3 AD12-3 battery voltage
// 4 AD12-4 R satellite static, also CVA-- Iw out
// 5 AD12-5 main static
// 6 AD12-6 L satellite static
// 7 AD12-7  freestream differential
// 8 AD12-8 L satellite freestream differential
// 9 AD12-9 main probe differential
// 10 AD12-10 L satellite probe differential
////////////// DEFINITIONS FOR -AV3b (through BLDS-G Dec 2009) HARDWARE CONFIGURATION //////////////
// ... PA & RB digital inputs/outputs (PA0-7 map to 0-7 and RB0-7 map to 16 to 23)
// 0 PA0 i counter
// 1 PA1 o turn on 3 pressure sensors, LED, bat voltage divider, temp sensor
// 2 PA2 o stage power
// 3 PA3 o satellite R power (also used for Conrad power)
// 4 PA4 o stage direction AND rotary step
// 5 PA5 i rotary home if ON
// 6 PA6 i stage lower limit
// 7 PA7 i stage upper limit
// 16 RB0
// 17 RB1 o satellite L power
```

```
// 18 RB2
// 19 RB3 o rotary power... also, rotary home encoder power
// 20 RB4
// 21 RB5 o encoder power... MUST be ON to count encoder pulses!
// 22 RB6
// 23 RB7 o stage extra power (smaller resistor for current limit)
// ... AD12 12-bit A/D inputs (read with CHAN(**), ** = 0-10 (11 channels))
// 0 AD12-0 main static
// 1 AD12-1 R satellite probe differential
// 2 AD12-2 R satellite freestream differential
// 3 AD12-3 battery voltage
// 4 AD12-4 R satellite static
// 5 AD12-5 main freestream differential
// 6 AD12-6 L satellite static
// 7 AD12-7 temperature
// 8 AD12-8 L satellite freestream differential
// 9 AD12-9 main probe differential
// 10 AD12-10 L satellite probe differential

// Enter startup subroutines here
        print
        print "*** BLDS_GT_7x10_v1_2013-07-01  JUL 2013 ***"
        gosub startup
        gosub batt_check
        batvfs$ = str(#7.3F, batvf!, ",")
        print "Battery voltage: " , batvfs$, nl$

        store #$, "Program Version: BLDS_GT_7x10_v1_2013-01-29 ", nl$
        gosub input_time       // Time Initialization
        gosub mainmenu
        gosub shutdown
mainmenu:
        choice1$ = "N"
        while (choice1$<> "3") // loop main menu until user exits
                gosub printmainmenu
                input "Menu Choice: " choice1$, #1
                if (choice1$ = "1")
                        gosub taskmenu
                else
                        if (choice1$ = "2")
                                gosub generalnoteinput
                        else
                                if (choice1$ = "3")
                                        print "Exiting..."
                                else
                                        print nl$, "Invalid choice! Please re-enter."
```

```
                        endif
                    endif
                endif
        wend
        // while loop terminated, Menu choice "3" was entered by user
return
printmainmenu:
        print nl$, "Main Menu"
        print " 1) Task Menu"
        print " 2) Add General Notes"
        print " 3) Exit Program"
return
generalnoteinput:
        answer1$ = "N"
        while (answer1$ <> "Y")
                // 1 byte = 1 character
                input "Enter General Notes (limit 80 characters): " generalnote$, #80
                print "You entered: ", generalnote$
                input "Is this correct? <Y or N>: " answer1$, #1
        wend
        // while loop terminates when user enters Y
        store #$, "General Notes: ", generalnote$, nl$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0
return
taskmenu:
        choice2$ = "N"
        while (choice2$ <> "7") // loop task menu until user exits
                gosub printtaskmenu
                input "Menu Choice: " choice2$, #1
                if (choice2$ = "1")
                        gosub batterycheck
                else
                        if (choice2$ = "2")
                                gosub freestream
                        else
                                if (choice2$ = "3")
                                        gosub windoff
                                else
                                        if (choice2$ = "4")
                                                gosub hotwireprofile
                                        else
                                                if (choice2$ = "5")
                                                        gosub hotwirecalibration
                                                else
                                                        if (choice2$ = "6")
```

```
                                                    gosub specificnotes
                                                else
                                                    if (choice2$ = "7")
                                                        print "Exiting task
menu..."
                                                    else
                                                        print nl$, "Invalid
Choice! Please re-enter."
                                                    endif
                                                endif
                                            endif
                                        endif
                                    endif
                                endif
                            endif
        wend
return
printtaskmenu:
        print nl$, "Task Menu"
        print " 1) Battery Check (stored)"
        print " 2) Freestream Check for 10 seconds (NOT stored)"
        print " 3) Take Pressure Windoff Data for 10 seconds (stored)"
        print " 4) Take Hot-Wire Profile (stored)"
        print " 5) Hot-Wire Calibration (stored)"
        print " 6) Notes (stored)"
        print " 7) Exit Task Menu"
return
batterycheck:
        print "Battery Check (stored)", nl$
        gosub batt_check
        batvfs$ = str(#7.3F, batvf!, ",")
        print "Battery voltage: " , batvfs$, nl$
        store #$, "Battery Voltage = "
        store #$,batvfs$,nl$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0
return
freestream:
        print "Freestream Data (NOT stored)", nl$
        //DATA ACQUISITION
        //turn ON power to main sensors and slight warm-up delay
        pset 17
        sleep 0
        sleep 20
        // from user get rate to test sensors
        //read & average main sensor
```

```
        pitotsignal! = 0
        pitotave! = 0
        staticsignal! = 0
        staticave! = 0
        Te! = 0
        tempave! = 0
        batv! = 0
        batvave! = 0
        duration = 10  // seconds
        starttime = ?    // Start Max Wait Timer
        endtime = ?     // endtime used to calculate elapsed time
        k = 1              // number of iterations so far (used in running ave calculation)
        while ( endtime - starttime < duration )
                kk! = k
                pitotsignal! = chan(7)
                pitotave! = pitotave! + ((pitotsignal! - pitotave!) / kk!)
                staticsignal! = chan(5)
                staticave! = staticave! + ((staticsignal! - staticave!) / kk!)
                Te! = chan(0)
                tempave! = tempave! + ((Te! - tempave!) / kk!)
                batv! = chan(3)
                batvave! = batvave! + ((batv! - batvave!) / kk!)
                k = k + 1         // increment our iteration counter
                endtime = ?     // capture the current time
        wend
        //turn off power to main sensors
        pclr 17
        //compute sensor avg voltages, battery avg voltages, & temp in C
        pitot! = (pitotave! * .0000763)
        static! = (staticave! * .0000763)
        T! = (tempave! * .00763) - 273.0       // 100.0 * (tempave! * 0.0000763) - 273.0
        batvf! = (batvave! * 3 * .0000763)     // 1/3 divider circuit in -AV3
        readrtc
        rtime
        times$ = str(#02, ?(5),"/",?(4),"/",?(3)," ",?(2),":",?(1),":",?(0), ",")
        statics$ = str(#7.4F, static!, ",")
        pitots$ = str(#7.4F, pitot!, ",")
        temps$ = str(#7.2F, T!, ",")
        batvfs$ = str(#7.3F, batvf!, ",")
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0
        //PRINT TO SCREEN (for benchtop monitor/checkout)
        print "Time,MPitot(V),Static(V),Temp(C),BatVolt,", nl$
        print times$, pitots$, statics$, temps$, batvfs$, nl$
return
windoff:
```

```
store #$, "Wind-Off Data", nl$
print "Wind-Off Data (stored)", nl$
//DATA ACQUISITION
//turn ON power to main sensors and slight warm-up delay
pset 17
sleep 0
sleep 20
//read & average main sensor
pitotsignal! = 0
pitotave! = 0
staticsignal! = 0
staticave! = 0
prestonsignal! = 0
prestonave! = 0
Te! = 0
tempave! = 0
batv! = 0
batvave! = 0
// Take data for 10 seconds
duration = 10  // seconds
starttime = ?    // Start Max Wait Timer
endtime = ?     // endtime used to calculate elapsed time
k = 0              // number of iterations so far (used in running ave calculation)
while ( endtime - starttime < duration )
        k = k + 1         // increment our iteration counter
        kk! = k
        pitotsignal! = chan(7)
        pitotave! = pitotave! + ((pitotsignal! - pitotave!) / kk!)
        staticsignal! = chan(5)
        staticave! = staticave! + ((staticsignal! - staticave!) / kk!)
        prestonsignal! = chan(9)
        prestonave! = prestonave! + ((prestonsignal! - prestonave!) / kk!)
        Te! = chan(0)
        tempave! = tempave! + ((Te! - tempave!) / kk!)
        batv! = chan(3)
        batvave! = batvave! + ((batv! - batvave!) / kk!)
        endtime = ?     // capture the current time
wend
//turn off power to main sensors
pclr 17
//compute sensor avg voltages, battery avg voltages, & temp in C
pitot! = (pitotave! * .0000763)
static! = (staticave! * .0000763)
preston! = (prestonave! * .0000763)
T! = (tempave! * .00763) - 273.0       // 100.0 * (tempave! * 0.0000763) - 273.0
batvf! = (batvave! * 3 * .0000763)     // 1/3 divider circuit in -AV3
```

```
        readrtc
        rtime
        //DATA STORAGE
        store #$, "Time,MPitot(V),Static(V),Preston(V),Temp(C),BatVolt,Samples", nl$
        times$ = str(#02, ?(5),"/",?(4),"/",?(3)," ",?(2),":",?(1),":",?(0), ",")
        store #$, times$
        pitots$ = str(#7.4F, pitot!, ",")
        store #$, pitots$
        statics$ = str(#7.4F, static!, ",")
        store #$, statics$
        prestons$ = str(#7.4F, preston!, ",")
        store #$, prestons$
        temps$ = str(#7.2F, T!, ",")
        store #$, temps$
        batvfs$ = str(#7.3F, batvf!, ",")
        store #$, batvfs$
        samples$ = str(" ", k, ",", nl$)
        store #$, samples$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0

        //PRINT TO SCREEN (for benchtop monitor/checkout)
        print "Time,MPitot(V),Static(V),Preston(V),Temperature(C),Voltage,Samples",
nl$
        print times$, pitots$, statics$, prestons$, temps$, batvfs$, samples$, nl$
return
hotwireprofile:
        store #$, "Hot-Wire Profile Data", nl$
        print "Hot-Wire Data (stored)", nl$
        gosub noteinputmenu
        for ipro = 1 to npros
                // initialize profile variables
                iypt = 0
                totalcount = 0
                target = r
                // ensure stage is at lower limit
                gosub lochk
                // print header for current profile
                gosub profile_header
                while (totalcount < maxtotal)
                        gosub take_pt_data
                        // LO-BATTERY CHECK
                        if (batvf! < lowbatvalue!)      // less than 5.5 Volts, do thorough
check
                                gosub batt_check
                        endif
```

```
                        // move stage to next ypt
                        gosub next_stage_pt
                        if (badmove = 1)                        // if stage fails to
move...
                                totalcount = maxtotal           // exit while loop early
                                ipro = ipro - 1                 // redo current profile
                        endif
                wend
                // move stage all the way down for next profile or finish
                gosub stage_down
        next ipro
return
hotwirecalibration:
        store #$, "Calibration Data", nl$
        print "Hot-Wire Calibration (stored)", nl$
        CVAvwout = 2
        CVAiwout = 4
        CVAvwin = 1
        // initialize sensor variables
        vwoutsignal! = 0
        vwoutave! = 0
        iwoutsignal! = 0
        iwoutave! = 0
        vwinsignal! = 0
        vwinave! = 0
        pitotsignal! = 0
        pitotave! = 0
        staticsignal! = 0
        staticave! = 0
        Te! = 0
        tempave! = 0
        batv! = 0
        batvave! = 0
        answer$ = "N"
        while (answer$ <> "Y")
                input "What V_w would you like to calibrate at?: " vwireset!
                print "You entered ", vwireset!
                input "Is this correct? <Y or N>: " answer$, #1
        wend
        pset 7
        sleep 0
        sleep 20
        vwiresets$ = str(#7.3F, vwireset!)
        store #$, "V_w = ", vwiresets$, nl$
        vwinsignal! = chan(1)
        vwinsignal! = (vwinsignal! * .0000763 / 5)
```

```
vwireset1! = vwireset! - 0.016
if (vwireset1! >= vwinsignal!)
        while(vwireset1! > vwinsignal!)
                pset 3
                pset 1
                pclr 1
                pclr 3
                vwinsignal! = chan(1)
                vwinsignal! = (vwinsignal! * .0000763 / 5)
        wend
else
        if (vwireset! < vwinsignal!)
                pset 1
                pset 3
                pclr 1
                vwinsignal! = chan(1)
                vwinsignal! = (vwinsignal! * .0000763 / 5)
                while(vwireset! <= vwinsignal!)
                        pset 1
                        pclr 1
                        vwinsignal! = chan(1)
                        vwinsignal! = (vwinsignal! * .0000763 / 5)
                wend
        endif
endif
pclr 1, 3
vwoutsignal! = 0
vwoutave! = 0
iwoutsignal! = 0
iwoutave! = 0
vwinsignal! = 0
vwinave! = 0
pitotsignal! = 0
pitotave! = 0
staticsignal! = 0
staticave! = 0
Te! = 0
tempave! = 0
batv! = 0
batvave! = 0
answer4$ = "N"
while (answer4$ <> "Y")
        // 1 byte = 1 character
        input "Enter calibration notes (limit 80 characters): " note$, #80
        print "You entered: ", note$
        input "Is this correct <Y or N>: " answer4$, #1
```

```
        wend
        store #$, "Notes: ", note$, nl$
        duration = 10  // seconds
        starttime = ?    // Start Max Wait Timer
        endtime = ?     // endtime used to calculate elapsed time
        k = 0              // number of iterations so far (used in running ave calculation)
        pset 5, 17
        sleep 0
        sleep 20
        while ( endtime - starttime < duration )
                // read & average main sensors
                k = k + 1          // increment our iteration counter
                kk! = k
                pitotsignal! = chan(7)
                pitotave! = pitotave! + ((pitotsignal! - pitotave!) / kk!)
                staticsignal! = chan(5)
                staticave! = staticave! + ((staticsignal! - staticave!) / kk!)
                Te! = chan(0)
                tempave! = tempave! + ((Te! - tempave!) / kk!)
                batv! = chan(3)
                batvave! = batvave! + ((batv! - batvave!) / kk!)
                vwoutsignal! = chan(CVAvwout)
                vwoutave! = vwoutave! + ((vwoutsignal! - vwoutave!) / kk!)
                iwoutsignal! = chan(CVAiwout)
                iwoutave! = iwoutave! + ((iwoutsignal! - iwoutave!) / kk!)
                vwinsignal! = chan(CVAvwin)
                vwinave! = vwinave! + ((vwinsignal! - vwinave!) / kk!)
                endtime = ?     // capture the current time
        wend
        pclr 5, 17
        // compute sensor avg voltages, battery avg voltages, & temp in C
        pitot! = (pitotave! * .0000763)
        static! = (staticave! * .0000763)
        T! = (tempave! * .00763) - 273.0       // 100.0 * (tempave! * 0.0000763) - 273.0
        batvf! = (batvave! * 3 * .0000763)     // 1/3 divider circuit in -AV3
        vwout! = (vwoutave! * .0000763 / 5)
        iwout! = (iwoutave! * .0000763)
        vwin! = (vwinave! * .0000763 / 5)
        store #$,
"MPitot(V),Static(V),Temperature(C),Voltage,V_wout(V),I_wout(V),V_win(V),Samples
", nl$
        print nl$,
"MPitot(V),Static(V),Temperature(C),Voltage,V_wout(V),I_wout(V),V_win(V),Samples
", nl$
        pitots$ = str(#7.4F, pitot!, ",")
        store #$, pitots$
```

```
        statics$ = str(#7.4F, static!, ",")
        store #$, statics$
        temps$ = str(#7.2F, T!, ",")
        store #$, temps$
        batvfs$ = str(#7.3F, batvf!, ",")
        store #$, batvfs$
        vwouts$ = str(#7.4F, vwout!, ",")
        store #$, vwouts$
        iwouts$ = str(#7.4F, iwout!, ",")
        store #$, iwouts$
        vwins$ = str(#7.4F, vwin!, ",")
        store #$, vwins$
        samples$ = str(" ", k, ",", nl$)
        store #$, samples$

        print pitots$, statics$, temps$, batvfs$, vwouts$, iwouts$, vwins$, samples$, nl$
        pclr 1, 3, 5, 7, 17
return

noteinputmenu:
        answer3$ = "N"
        while(answer3$ <> "Y")
                print
                input "Seconds to wait before each profile: " profileWait
                input "Number of seconds to read data for: " datatime
                input "Number of counts to move near wall: " r
                input "Muliplier once off wall: " m!
                input "Max step increment, in encoder counts: " maxcount
                input "Number of profiles: " npros
                input "Maximum total encoder counts (12800 per inch for 16:1/51200 per
inch for 64:1): " maxtotal
                gosub suggest_vw
                input "V_w? (Will round to the LOWER multiple of 0.016V): " vwireset!
                print nl$, "Seconds to wait before each profile: ", profileWait
                print "Number of seconds to read data for: ", datatime
                print "Number of counts to move near wall: ", r
                print "Multiplier once off wall: ", #10.3F , m!
                print "Max step increment in encoder counts is: ", maxcount
                print "Total number of profiles taken will be: ", npros
                print "The maximum number of counts will be: ", maxtotal
                print "The V_w for the following boundary layer profiles will be: ",
vwireset!
                print "Does this data look correct?"
                input "Input Y to continue, any other command will allow you to redo
inputs: " answer3$, #1
        wend
```

```
        // while loop terminates when user enters Y
        store #$, "Seconds to wait before each profile: "
        profileWaits$ = str(profileWait, nl$)
        store #$, profileWaits$
        store #$, "Number of seconds to read data for: "
        datatimes$ = str(datatime, nl$)
        store #$, datatimes$
        store #$, "Number of counts to move near wall: "
        rs$ = str(r, nl$)
        store #$, rs$
        store #$, "Muliplier once off wall: "
        mults$ = str(#7.3F, m!, nl$)
        store #$, mults$
        store #$, "Max step increment, in encoder counts: "
        maxcounts$ = str(maxcount, nl$)
        store #$, maxcounts$
        store #$, "Number of profiles: "
        npross$ = str(npros, nl$)
        store #$, npross$
        store #$, "Maximum total encoder counts (12800 per inch): "
        maxtotals$ = str(maxtotal, nl$)
        store #$, maxtotals$
        store #$, "V_w: "
        vwiresets$ = str(#7.3F, vwireset!, nl$)
        store #$, vwiresets$
return
specificnotes:
        answer4$ = "N"
        while(answer4$ <> "Y")
                // 1 byte = 1 character
                input "Enter Additional Notes (limit 80 characters): " note$, #80
                print "You entered: ", note$
                input "Is this correct <Y or N>: " answer4$, #1
        wend
        // while loop terminates when user enters Y
        store #$, "Notes: ", note$, nl$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0
return
suggest_vw:
        pset 17
        Te! = 0
        input "What is the cold resistance (in ohms) of the hot-wire at 20 degree Celsius?
" R_inf!
        input "What is the maximum speed (in m/s) the hot-wire will experience during
the gathering of boundary layer profile data? " U!
```

```
        input "What is the ambient pressure (in mm Hg)? " P_inf!
        Ohr! = 2
        alpha_20! = 0.0042
        dia! = 0.0000038                        // meters
        L! = 0.00127                            // meters
        U! = U! * 0.1
        Te! = chan(0)
        T_inf! = (Te! * .00763)                 // Kelvin
        pclr 17
        R_w! = Ohr! * R_inf!
        T_w! = T_inf! + (((R_w!/R_inf!) - 1)/alpha_20!)
        T_f! = (T_w! + T_inf!)/2
        k_f! = 0.0239 + 0.000073189 * (T_f!-273.0) - 0.000000018982 * (T_f!-
273.0)*(T_f!-273.0)
        T_ratio! = T_f!/T_inf!
        rho_f! = 1.2929 * (273.0/T_f!) * (P_inf!/760)
        nu_f! = (84.986+(7 * T_f!) - 0.0037501 * (T_f! * T_f!)) * 0.00000001/rho_f!
        Re! = U!*dia!/nu_f!
        // Approximation of Re^0.45
        approx! = 1 + 0.45*log(Re!) + ((0.45*log(Re!)) * (0.45*log(Re!)))/2 +
((0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)))/6 + ((0.45*log(Re!)) *
(0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)))/24 + ((0.45*log(Re!)) *
(0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)))/120 +
((0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!)) * (0.45*log(Re!))
* (0.45*log(Re!)))/720
        // Approximation of (T_f/T_inf)^0.17
        T_rat_approx! = 1 + 0.17*log(T_ratio!) + ((0.17*log(T_ratio!)) *
(0.17*log(T_ratio!)))/2 + ((0.17*log(T_ratio!)) * (0.17*log(T_ratio!)) *
(0.17*log(T_ratio!)))/6
        A_s! = 3.14159 * dia! * L!
        V_w! = sqr((0.24 + 0.56 * approx!) * T_rat_approx! * k_f! * A_s! * ((Ohr!-
1)/alpha_20!) * Ohr! * R_inf!/dia!)
        V_ws$ = str(#7.3F, V_w!, nl$)
        print nl$, "The suggested wire voltage to set the hot-wire to is: ", V_ws$
return
///////////////// STARTUP SUBROUTINE //////////////////////
startup:
        pclr 16, 17, 18, 19, 20, 21, 22, 23     // Make sure no power is applied to all main
board/satellite outputs/ all logic 0 (false)
        pclr 1, 3, 5, 7                         // Power off and logic 0 (false) for all CVA-
related outputs
        cbreak quick_exit                       // goto quick_exit label when Ctrl-C pressed
        error = 0
        onerr quick_exit, error                 // goto quick_exit if execution error occurs,
                                                // error information stored in 'error'
        lowbatvalue! = 5.3                      // Initialize low-battery threshhold
```

```
        nl$ = str(\13,\10)                      // a new-line string for formatting output
        if (DFERASED = 0)
                print nl$, "WARNING: data file not erased"
                print "Enter Y to continue, otherwise program will terminate"
                input answer$,#1
                if (answer$ <> "Y")
                        stop
                endif
        endif
return
////////////////////////////////////////////////////////
/////////////// QUICK_EXIT SUBROUTINE ////////////////////
// Control-C was pressed, OR error occured, exit gracefully
// Error List on page 144 of TFBasic Manual
quick_exit:
        pclr 16, 17, 18, 19, 20, 21, 22, 23     // Make sure no power is applied to all main
board/satellite outputs/ all logic 0 (false)
        pclr 1, 3, 5, 7                         // Power off and logic 0 (false) for all CVA-
related outputs
        count                                   // Make sure background counter is off
        call &hFD88,0                           // flush output buffer to datafile
        if (error = 0)                          // error did not occur
                print "Control-C Pressed, Program Stopped, Ready for Data Offload."
        else                                    // error did occur
                message$ = str("Error #", error / 65536, "  @", #05H, error % 65536)
                print message$
                store #$, message$, nl$
                call &hFD88,0                   // flush output buffer to datafile
        endif
        stop
return                                          // Program will never reach this return
statement
////////////////////////////////////////////////////////
//////////////////BATT_CHECK SUBROUTINE /////////////////
batt_check:
        pset 17                                 // turn on power to main sensors & batt
check circuit
        sleep 0                                 // slight warm up delay
        sleep 20
        batv! = 0
        batvave! = 0
        batvf! = 0
        for k = 1 to 50                         // 50 samples to be sure
                sleep 0
                sleep 5
                kk! = k
```

```
                batv! = chan(3)
                batvave! = batvave! + ((batv! - batvave!) / kk!)
        next k
        pclr 17                                         // turn off power to main sensors
        batvf! = (batvave! * 3 * .0000763)              // 1/3 divider circuit in -AV3
        if (batvf! < lowbatvalue!)
                message$ = str("BATTERY < ", #.3F, lowbatvalue!, " V, SHUTTING
DOWN, V = ", #.3F, batvf!)
                store #$, message$, nl$
                print nl$, message$
                call &hFD88,0           // flush data buffer
                gosub stage_down        // move stage down
                gosub shutdown
        endif
return
////////////////////////////////////////////////////////
/////////////// INPUT_TIME SUBROUTINE ////////////////////
input_time:
        answer$ = "N"
        while (answer$ <> "Y")                  // re-input time until user enters "Y"
                print
                input "Year:   "?(5)
                input "Month:  "?(4)
                input "Day:    "?(3)
                input "Hour:   "?(2)
                input "Minute: "?(1)
                input "Second: "?(0)
                stime
                setrtc
                times$ = str(#02, ?(4),"/",?(3),"/",?(5)," ",?(2),":",?(1),":",?(0))
                print "The time stamp is: ", times$
                print "Does this data look correct?"
                input "Input Y to continue, any other command will allow you to redo
time: " answer$, #1
        wend
return
////////////////////////////////////////////////////////
/////////////// NEXT_STAGE_PT SUBROUTINE ///////////////
next_stage_pt:
        badmove = 0                     // badmove gets set to 1 if stage fails to move
        oldtar = target
        if (iypt > 8)                   // the wall is 8 ypts long
                target = oldtar * m!
        endif
        if (target > maxcount)
                target = maxcount
```

```
        endif
        pset 18                         //turn ON stage encoder power
        sleep 0
        sleep 20
        // turn on background counting process to count encoder pulses
        backcount = 0
        count backcount
        readrtc                         // copy PIC time to ?-variable
        starttime = ?
// PWM parameters--added 1/30/2013 for v2 to allow smaller steps
// on_time is PWM Duty Cycle as an integer percentage"
// where 10 = 10% and 100 = 100%"
        on_time =  7
        off_time = 100 - on_time
        pclr 19                         // set direction UP
//      pset 16                         // turn ON stage motor power
        while (backcount < target)
// PWM loop ...
                pset 16 // stage pwr on
                for i = 1 to on_time
                next i
                pclr 16 // stage pwr off
                for i = 1 to off_time
                next i
        wend
        pclr 16         // turn OFF stage motor power
        sleep 0         // give stage time to come to a stop
        sleep 50
        count           // turn off background counting
        pclr 18         // turn OFF encoder power to limit battery power drain
        // done with stage move, update encoder total counts
        totalcount = totalcount + backcount
        // done with stage move, now at next ypt
        iypt = iypt + 1
return
////////////////////////////////////////////////////////
///////////////// LOCHK SUBROUTINE ////////////////////////
// Ensure stage is at lower limit to start profile
// Check to see if stage is at lower limit, if not,
// try to drive twice more at most
// lower limit reached if PA4=0 when RB3=1
// upper limit reached if PA2=0 when RB3=0
lochk:
        pset 19
        lolim = pin(4)
        pclr 19
```

```
        attempt = 1
        while (lolim <> 0 & attempt <= 8)
                message$ = str("lochk: Low limit not reached, lolim = ", lolim)
                print message$                          // for benchtop testing
                store #$, message$, nl$                 // store message to datafile
                call &hFD88,0                           // flush datafile buffer
                gosub stage_down                        // redrive stage
                pset(19)
                lolim = pin(4)                          // low limit reached if lolim = 0
                pclr(19)
                attempt = attempt + 1           // increment number of redrive attempts
        wend
return
////////////////////////////////////////////////////////
///////////////// PROFILE_HEADER SUBROUTINE ////////////////
profile_header:
        //store and print header data for current profile
        readrtc
        rtime
        times$ = str(#02, ?(5),"/",?(4),"/",?(3)," ",?(2),":",?(1),":",?(0), ",")
        store #$, times$
        ipros$ = str(" profile no, ", ipro, ", ")
        store #$, ipros$
        lolims$ = str("lolim if 0, ", lolim, nl$)
        store #$, lolims$
        store #$, "Time,MPitot(V),Static(V),Temperature(C),Voltage,Encoder
totcnt,V_wout(V),I_wout(V),V_win(V),SDE Average (V), Samples", nl$
        print nl$, times$, ipros$, lolims$ ;
        print nl$, "Time,MPitot(V),Static(V),Temperature(C),Voltage,Encoder
totcnt,V_wout(V),I_wout(V),V_win(V),SDE Average(V), Samples", nl$
return
////////////////////////////////////////////////////////
///////////////// STAGE_DOWN SUBROUTINE ////////////////
stage_down:
        // move stage DOWN all the way
        pset 19
        pset 16
        sleep 0
        sleep 1000
        pclr 16
        pclr 19
return
////////////////////////////////////////////////////////
///////////////// SHUTDOWN SUBROUTINE //////////////////
shutdown:
        //CLEAR OUTPUTS, just to be sure
```

```
        pclr 16, 17, 18, 19, 20, 21, 22, 23      // Make sure no power is applied to all main
board/satellite outputs/ all logic 0 (false)
        pclr 1, 3, 5, 7                          // Power off and logic 0 (false) for all CVA-
related outputs
        stop                                     // go ahead and stop, power-on but in lo-
power config
return
////////////////////////////////////////////////////
/////////////// TAKE_PT_DATA SUBROUTINE ////////////////
take_pt_data:
        //DATA ACQUISITION
        //pause 5 seconds after move
        sleep 0
        sleep 500
        //turn ON power to main sensors and slight warm-up delay
        pset 17, 5, 7
        sleep 0
        sleep 20
        //read & average main sensor
        pitotsignal! = 0
        pitotave! = 0
        staticsignal! = 0
        staticave! = 0
        Te! = 0
        tempave! = 0
        batv! = 0
        batvave! = 0
        vwoutsignal! = 0
        vwoutave! = 0
        iwoutsignal! = 0
        iwoutave! = 0
        vwinsignal! = 0
        vwinave! = 0
        iwoutsdevave! = 0
        iwoutsumOfSquaresAve! = 0
        vwinsignal! = chan(1)
        vwinsignal! = (vwinsignal! * .0000763 / 5)
        vwireset1! = vwireset! - 0.016
        if (vwireset1! >= vwinsignal!)
                while(vwireset1! > vwinsignal!)
                        pset 3
                        pset 1
                        pclr 1
                        pclr 3
                        vwinsignal! = chan(1)
                        vwinsignal! = (vwinsignal! * .0000763 / 5)
```

```
                wend
        else
                if (vwireset! < vwinsignal!)
                        pset 1
                        pset 3
                        pclr 1
                        vwinsignal! = chan(1)
                        vwinsignal! = (vwinsignal! * .0000763 / 5)
                        while(vwireset! <= vwinsignal!)
                                pset 1
                                pclr 1
                                vwinsignal! = chan(1)
                                vwinsignal! = (vwinsignal! * .0000763 / 5)
                        wend
                endif
        endif
        duration = datatime     // seconds
        starttime = ?                   // Start Max Wait Timer
        endtime = ?                             // endtime used to calculate elapsed time
        k = 1                                   // number of iterations so far (used in
running ave calculation)
        while ( endtime - starttime < duration )
                k = k + 1                       // increment our iteration counter
                kk! = k
                pitotsignal! = chan(7)
                pitotave! = pitotave! + ((pitotsignal! - pitotave!) / kk!)
                staticsignal! = chan(5)
                staticave! = staticave! + ((staticsignal! - staticave!) / kk!)
                Te! = chan(0)
                tempave! = tempave! + ((Te! - tempave!) / kk!)
                batv! = chan(3)
                batvave! = batvave! + ((batv! - batvave!) / kk!)
                vwoutsignal! = chan(2)
                vwoutave! = vwoutave! + ((vwoutsignal! - vwoutave!) / kk!)
                iwoutsignal! = chan(4)
                iwoutave! = iwoutave! + ((iwoutsignal! - iwoutave!) / kk!)
                vwinsignal! = chan(1)
                vwinave! = vwinave! + ((vwinsignal! - vwinave!) / kk!)
                iwoutsumOfSquaresAve! = iwoutsumOfSquaresAve! + ((iwoutsignal! *
iwoutsignal! - iwoutsumOfSquaresAve!) / k)
                endtime = ?             // capture the current time
        wend
        iwoutsdevave! = sqr(iwoutsumOfSquaresAve! - (iwoutave! * iwoutave!))
        //turn off power to main sensors
        pclr 17, 1, 3, 5, 7
        //compute sensor avg voltages, battery avg voltages, & temp in C
```

```
        pitot! = (pitotave! * .0000763)
        static! = (staticave! * .0000763)
        T! = (tempave! * .00763) - 273.0                // 100.0 * (tempave! * 0.0000763) -
273.0
        batvf! = (batvave! * 3 * .0000763)              // 1/3 divider circuit in -AV3
        vwout! = (vwoutave! * .0000763 / 5)
        iwout! = (iwoutave! * .0000763)
        vwin! = (vwinave! * .0000763 / 5)
        iwoutsdevave! = (iwoutsdevave! * .0000763)
        readrtc
        rtime
        //DATA STORAGE
        times$ = str(#02, ?(5),"/",?(4),"/",?(3)," ",?(2),":",?(1),":",?(0), ",")
        store #$, times$
        pitots$ = str(#7.4F, pitot!, ",")
        store #$, pitots$
        statics$ = str(#7.4F, static!, ",")
        store #$, statics$
        temps$ = str(#7.2F, T!, ",")
        store #$, temps$
        batvfs$ = str(#7.3F, batvf!, ",")
        store #$, batvfs$
        encts$ = str(" ", totalcount, ",")
        store #$, encts$
        vwouts$ = str(#7.4F, vwout!, ",")
        store #$, vwouts$
        iwouts$ = str(#7.4F, iwout!, ",")
        store #$, iwouts$
        vwins$ = str(#7.4F, vwin!, ",")
        store #$, vwins$
        iwoutsdevaves$ = str(#7.5F, iwoutsdevave!, ",")
        store #$, iwoutsdevaves$
        samples$ = str(" ", k, ",", nl$)
        store #$, samples$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0
        //PRINT TO SCREEN (for benchtop monitor/checkout)
        print times$, pitots$, statics$, temps$, batvfs$, encts$, vwouts$, iwouts$, vwins$,
iwoutsdevaves$, samples$, nl$
return
////////////////////////////////////////////////////////
```

BLDS – GT Pressure Probe Program

```
// BLDS-GT-CVA NOV 2012/ JAN 2103
////////////// DEFINITIONS FOR  -GT-CVA (November 2012/ Jan 2013) HARDWARE CONFIGURATION //////////////
// ... PA & RB digital inputs/outputs (PA0-7 map to 0-7 and RB0-7 map to 16 to 23)
// 0 PA0 i counter
// 1 PA1 o CVA--Vw step
// 2 PA2 i stage upper limit
// 3 PA3 o CVA--Vw stop/set
// 4 PA4 i stage lower limit
// 5 PA5 o CVA-- sensor ON
// 6 PA6 i rotary home if ON
// 7 PA7 o CVA--power ON
// 16 RB0 o stage power
// 17 RB1 o turn on 3 pressure sensors, LED, bat voltage divider, temp sensor
// 18 RB2 o encoder power... MUST be ON to count encoder pulses!
// 19 RB3 o stage direction
// 20 RB4 o rotary step
// 21 RB5 o satellite R power (also used for Conrad power)
// 22 RB6 o satellite L power
// 23 RB7 o rotary power
// ... AD12 12-bit A/D inputs (read with CHAN(**), **=0-10 (11 channels)s
// 0 AD12-0 temperature
// 1 AD12-1 R satellite probe differential, also CVA-- Vw in check
// 2 AD12-2 R satellite freestream differential,  also CVA-- Vw out
// 3 AD12-3 battery voltage
// 4 AD12-4 R satellite static, also CVA-- Iw out
// 5 AD12-5 main static
// 6 AD12-6 L satellite static
// 7 AD12-7  freestream differential
// 8 AD12-8 L satellite freestream differential
// 9 AD12-9 main probe differential
// 10 AD12-10 L satellite probe differential
////////////// DEFINITIONS FOR -AV3b (through BLDS-G Dec 2009) HARDWARE CONFIGURATION //////////////
// ... PA & RB digital inputs/outputs (PA0-7 map to 0-7 and RB0-7 map to 16 to 23)
// 0 PA0 i counter
// 1 PA1 o turn on 3 pressure sensors, LED, bat voltage divider, temp sensor
// 2 PA2 o stage power
// 3 PA3 o satellite R power (also used for Conrad power)
// 4 PA4 o stage direction AND rotary step
// 5 PA5 i rotary home if ON
// 6 PA6 i stage lower limit
// 7 PA7 i stage upper limit
// 16 RB0
// 17 RB1 o satellite L power
```

```
// 18 RB2
// 19 RB3 o rotary power... also, rotary home encoder power
// 20 RB4
// 21 RB5 o encoder power... MUST be ON to count encoder pulses!
// 22 RB6
// 23 RB7 o stage extra power (smaller resistor for current limit)
// ... AD12 12-bit A/D inputs (read with CHAN(**), **=0-10 (11 channels))
// 0 AD12-0 main static
// 1 AD12-1 R satellite probe differential
// 2 AD12-2 R satellite freestream differential
// 3 AD12-3 battery voltage
// 4 AD12-4 R satellite static
// 5 AD12-5 main freestream differential
// 6 AD12-6 L satellite static
// 7 AD12-7 temperature
// 8 AD12-8 L satellite freestream differential
// 9 AD12-9 main probe differential
// 10 AD12-10 L satellite probe differential

// Enter startup subroutines here
        print
        print "*** BLDS_GT_7x10_v1_2013-07-01  JUL 2013 ***"
        gosub startup
        gosub batt_check
        batvfs$ = str(#7.3F, batvf!, ",")
        print "Battery voltage: " , batvfs$, nl$

        store #$, "Program Version: BLDS_GT_7x10_v1_2013-01-29 ", nl$
        gosub input_time       // Time Initialization
        gosub mainmenu
        gosub shutdown
mainmenu:
        choice1$ = "N"
        while (choice1$<> "3") // loop main menu until user exits
                gosub printmainmenu
                input "Menu Choice: " choice1$, #1
                if (choice1$ = "1")
                        gosub taskmenu
                else
                        if (choice1$ = "2")
                                gosub generalnoteinput
                        else
                                if (choice1$ = "3")
                                        print "Exiting..."
                                else
                                        print nl$, "Invalid choice! Please re-enter."
```

```
                                endif
                        endif
                endif
        wend
        // while loop terminated, Menu choice "3" was entered by user
return
printmainmenu:
        print nl$, "Main Menu"
        print " 1) Task Menu"
        print " 2) Add General Notes"
        print " 3) Exit Program"
return
generalnoteinput:
        answer1$ = "N"
        while (answer1$ <> "Y")
                // 1 byte = 1 character
                input "Enter General Notes (limit 80 characters): " generalnote$, #80
                print "You entered: ", generalnote$
                input "Is this correct? <Y or N>: " answer1$, #1
        wend
        // while loop terminates when user enters Y
        store #$, "General Notes: ", generalnote$, nl$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0
return
taskmenu:
        choice2$ = "N"
        while (choice2$ <> "7") // loop task menu until user exits
                gosub printtaskmenu
                input "Menu Choice: " choice2$, #1
                if (choice2$ = "1")
                        gosub batterycheck
                else
                        if (choice2$ = "2")
                                gosub freestream
                        else
                                if (choice2$ = "3")
                                        gosub windoff
                                else
                                        if (choice2$ = "4")
                                                gosub hotwireprofile
                                        else
                                                if (choice2$ = "5")
                                                        gosub hotwirecalibration
                                                else
                                                        if (choice2$ = "6")
```

```
                                                    gosub specificnotes
                                                else
                                                    if (choice2$ = "7")
                                                        print "Exiting task
menu..."
                                                    else
                                                        print nl$, "Invalid
Choice! Please re-enter."
                                                    endif
                                                endif
                                            endif
                                        endif
                                    endif
                                endif
                            endif
        wend
return
printtaskmenu:
        print nl$, "Task Menu"
        print " 1) Battery Check (stored)"
        print " 2) Freestream Check for 5 seconds (NOT stored)"
        print " 3) Take Pressure Windoff Data for 5 seconds (stored)"
        print " 4) Take Pressure Profile (stored)"
        print " 5) Hot-Wire Calibration (stored)"
        print " 6) Notes (stored)"
        print " 7) Exit Task Menu"
return
batterycheck:
        print "Battery Check (stored)", nl$
        gosub batt_check
        batvfs$ = str(#7.3F, batvf!, ",")
        print "Battery voltage: " , batvfs$, nl$
        store #$, "Battery Voltage = "
        store #$, batvfs$ ,nl$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0
return
freestream:
        print "FreeStream Data (NOT stored)", nl$
        //DATA ACQUISITION
        //turn ON power to main sensors and slight warm-up delay
        pset 17
        sleep 0
        sleep 20
        // from user get rate to test sensors
        answer$ = "N"
```

```
        while (answer$ <> "Y")
                input "Sensor read rate in reads/second: " readrate
                print "You entered ", readrate
                print "Input Y to continue, any other command will allow you to redo
input"
                input answer$, #1
        wend
        sleeptime = 1.0 / float(readrate) * 100  // will give sleep time in centiseconds
        iterations = readrate*5
        //read & average main sensor
        pitotsignal! = 0
        pitotave! = 0
        staticsignal! = 0
        staticave! = 0
        Te! = 0
        tempave! = 0
        batv! = 0
        batvave! = 0
        // Sample for 5 second
        for k = 1 to iterations
                sleep 0
                sleep sleeptime
                kk! = k
                pitotsignal! = chan(7)
                pitotave! = pitotave! + ((pitotsignal! - pitotave!) / kk!)
                staticsignal! = chan(5)
                staticave! = staticave! + ((staticsignal! - staticave!) / kk!)
                Te! = chan(0)
                tempave! = tempave! + ((Te! - tempave!) / kk!)
                batv! = chan(3)
                batvave! = batvave! + ((batv! - batvave!) / kk!)
        next k
        //turn off power to main sensors
        pclr 17
        //compute sensor avg voltages, battery avg voltages, & temp in C
        pitot! = (pitotave! * .0000763)
        static! = (staticave! * .0000763)
        T! = (tempave! * .00763) - 273.0       // 100.0 * (tempave! * 0.0000763) - 273.0
        batvf! = (batvave! * 3 * .0000763)     // 1/3 divider circuit in -AV3
        readrtc
        rtime
        times$ = str(#02, ?(5),"/",?(4),"/",?(3)," ",?(2),":",?(1),":",?(0), ",")
        statics$ = str(#7.3F, static!, ",")
        pitots$ = str(#7.3F, pitot!, ",")
        temps$ = str(#7.2F, T!, ",")
        batvfs$ = str(#7.3F, batvf!, ",")
```

```
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0

        //PRINT TO SCREEN (for benchtop monitor/checkout)
        print "Time,MPitot(V),Static(V),Temp(C),BatVolt,", nl$
        print times$, pitots$, statics$, temps$, batvfs$, nl$
return
windoff:
        store #$, "Wind-Off Data", nl$
        print "Wind-Off Data (stored)", nl$
        //DATA ACQUISITION
        //turn ON power to main sensors and slight warm-up delay
        pset 17
        sleep 0
        sleep 20
        // from user get rate to test sensors
        answer$ = "N"
        while (answer$ <> "Y")
                input "Sensor read rate in reads/second: " readrate
                print "You entered ", readrate
                print "Input Y to continue, any other command will allow you to redo
input"
                input answer$, #1
        wend
        sleeptime = 1.0 / float(readrate) * 100  // will give sleep time in centiseconds
        iterations = readrate*5
        //read & average main sensor
        pitotsignal! = 0
        pitotave! = 0
        staticsignal! = 0
        staticave! = 0
        prestonsignal! = 0
        prestonave! = 0
        Te! = 0
        tempave! = 0
        batv! = 0
        batvave! = 0
        // Take data for 5 seconds
        for k = 1 to iterations
                sleep 0
                sleep sleeptime
                kk! = k
                pitotsignal! = chan(7)
                pitotave! = pitotave! + ((pitotsignal! - pitotave!) / kk!)
                staticsignal! = chan(5)
                staticave! = staticave! + ((staticsignal! - staticave!) / kk!)
```

```
                prestonsignal! = chan(9)
                prestonave! = prestonave! + ((prestonsignal! - prestonave!) / kk!)
                Te! = chan(0)
                tempave! = tempave! + ((Te! - tempave!) / kk!)
                batv! = chan(3)
                batvave! = batvave! + ((batv! - batvave!) / kk!)
        next k
        //turn off power to main sensors
        pclr 17
        //compute sensor avg voltages, battery avg voltages, & temp in C
        pitot! = (pitotave! * .0000763)
        static! = (staticave! * .0000763)
        preston! = (prestonave! * .0000763)
        T! = (tempave! * .00763) - 273.0       // 100.0 * (tempave! * 0.0000763) - 273.0
        batvf! = (batvave! * 3 * .0000763)     // 1/3 divider circuit in -AV3
        readrtc
        rtime
        //DATA STORAGE
        store #$, "Time,MPitot(V),Static(V),Preston(V),Temp(C),BatVolt", nl$
        times$ = str(#02, ?(5),"/",?(4),"/",?(3)," ",?(2),":",?(1),":",?(0), ",")
        store #$, times$
        pitots$ = str(#7.3F, pitot!, ",")
        store #$, pitots$
        statics$ = str(#7.3F, static!, ",")
        store #$, statics$
        prestons$ = str(#7.3F, preston!, ",")
        store #$, prestons$
        temps$ = str(#7.2F, T!, ",")
        store #$, temps$
        batvfs$ = str(#7.3F, batvf!, ",", nl$)
        store #$, batvfs$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0

        //PRINT TO SCREEN (for benchtop monitor/checkout)
        print "Time,MPitot(V),Static(V),Preston(V),Temperature(K),Voltage, ", nl$
        print times$, pitots$, statics$, prestons$, temps$, batvfs$, nl$
return
hotwireprofile:
        store #$, "Pressure Profile Data", nl$
        print "Pressure Data (stored)", nl$
        choice3$ = "Y"
        gosub noteinputmenu
        while (choice3$ = "Y")
                for ipro = 1 to npros
                        // initialize profile variables
```

```
                        iypt = 0
                        totalcount = 0
                        target = r
                        // ensure stage is at lower limit
                        gosub lochk
                        // print header for current profile
                        gosub profile_header
                        while (totalcount < maxtotal)
                                gosub take_pt_data
                                // LO-BATTERY CHECK
                                if (batvf! < lowbatvalue!)      // less than 5.5 Volts, do
thorough check
                                        gosub batt_check
                                endif
                                // move stage to next ypt
                                gosub next_stage_pt
                                if (badmove = 1)                        // if stage fails to
move...
                                        totalcount = maxtotal           // exit while loop
early
                                        ipro = ipro - 1                 // redo current profile
                                endif
                        wend
                        // move stage all the way down for next profile or finish
                        gosub stage_down
                next ipro
                gosub do_over
                choice3$ = answer2$
        wend
        // While loop terminates when user enters a value that is not Y
return
hotwirecalibration:
        store #$, "Calibration Data", nl$
        print "Hot-Wire Calibration (stored)", nl$
        CVAvwout = 2
        CVAiwout = 4
        CVAvwin = 1
        pset 7
        sleep 0
        sleep 20
        // initialize sensor variables
        vwoutsignal! = 0
        vwoutave! = 0
        iwoutsignal! = 0
        iwoutave! = 0
        vwinsignal! = 0
```

```
vwinave! = 0
input "What V_w would you like to calibrate at?: " vwireset!
vwiresets$ = str(#7.3F, vwireset!)
store #$, "V_w = ", vwiresets$, nl$
vwinsignal! = chan(1)
vwinsignal! = (vwinsignal! * .0000763 / 5)
vwireset1! = vwireset! - 0.016
if (vwireset1! >= vwinsignal!)
        while(vwireset1! > vwinsignal!)
                pset 3
                pset 1
                pclr 1
                pclr 3
                vwinsignal! = chan(1)
                vwinsignal! = (vwinsignal! * .0000763 / 5)
                print vwinsignal!
        wend
else
        if (vwireset! < vwinsignal!)
                pset 1
                pset 3
                pclr 1
                vwinsignal! = chan(1)
                vwinsignal! = (vwinsignal! * .0000763 / 5)
                while(vwireset! <= vwinsignal!)
                        pset 1
                        pclr 1
                        vwinsignal! = chan(1)
                        vwinsignal! = (vwinsignal! * .0000763 / 5)
                wend
        endif
endif
pclr 1, 3
answer$ = "Y"
while (answer$ <> "N")
        answer4$ = "N"
        while (answer4$ <> "Y")
                // 1 byte = 1 character
                input "Enter calibration notes (limit 80 characters): " note$, #80
                print "You entered: ", note$
                input "Is this correct <Y or N>: " answer4$, #1
        wend
        store #$, "Notes: ", note$, nl$
        duration = 5    // seconds
        starttime = ?   // Start Max Wait Timer
        endtime = ?     // endtime used to calculate elapsed time
```

```
                k = 1               // number of iterations so far (used in running ave
calculation)
                pset 5
                sleep 0
                sleep 20
                while ( endtime - starttime < duration )
                        // read & average main sensors
                        kk! = k
                        vwoutsignal! = chan(CVAvwout)
                        vwoutave! = vwoutave! + ((vwoutsignal! - vwoutave!) / kk!)
                        iwoutsignal! = chan(CVAiwout)
                        iwoutave! = iwoutave! + ((iwoutsignal! - iwoutave!) / kk!)
                        vwsignal! = chan(CVAvwin)
                        vwinave! = vwinave! + ((vwsignal! - vwinave!) / kk!)
                        k = k + 1                   // increment our iteration counter
                        endtime = ?                 // capture the current time
                wend
                pclr 5
                // compute sensor avg voltages, battery avg voltages, & temp in C
                vwout! = (vwoutave! * .0000763 / 5)
                iwout! = (iwoutave! * .0000763 * 24)
                vwin! = (vwinave! * .0000763 / 5)
                // print data to screen
                print
                print "V_wout (V) = ", vwout!
                print "I_wout (mA) = ", iwout!
                print "V_win  (V) = ", vwin!
                vwouts$ = str(#7.3F, vwout!, ",")
                iwouts$ = str(#7.3F, iwout!, ",")
                vwins$ = str(#7.3F, vwin!, ",", nl$)
                store #$, "V_wout(V),I_wout(mA),V_win(V)", nl$
                store #$, vwouts$
                store #$, iwouts$
                store #$, vwins$
                input "Would you like to take more calibration data <Y or N>: " answer$,
#1
        wend
        store #$, "End Calibration Data", nl$
        pclr 1, 3, 5, 7
return

noteinputmenu:
        answer3$ = "N"
        while(answer3$ <> "Y")
                print
                input "Seconds to wait before each profile: " profileWait
```

```
                input "Number of reads/second: " readrate
                input "Number of seconds to read data for: " datatime
                input "Number of counts to move near wall: " r
                input "Muliplier once off wall: " m!
                input "Max step increment, in encoder counts: " maxcount
                input "Number of profiles: " npros
                input "Maximum total encoder counts (12800 per inch for 16:1/51200 per
inch for 64:1): " maxtotal
                print nl$, "Seconds to wait before each profile: ", profileWait
                print "Number of reads/second: ", readrate
                print "Number of seconds to read data for: ", datatime
                print "Number of counts to move near wall: ", r
                print "Multiplier once off wall: ", #10.3F , m!
                print "Max step increment in encoder counts is: ", maxcount
                print "Total number of profiles taken will be: ", npros
                print "The maximum number of counts will be: ", maxtotal
                print "Does this data look correct?"
                input "Input Y to continue, any other command will allow you to redo
inputs: " answer3$, #1
        wend
        // while loop terminates when user enters Y
        sleeptime = 1.0/float(readrate) * 100
        iterations = readrate*datatime
        store #$, "Seconds to wait before each profile: "
        profileWaits$ = str(profileWait, nl$)
        store #$, profileWaits$
        store #$, "Number of reads/second: "
        readrates$ = str(readrate, nl$)
        store #$, readrates$
        store #$, "Number of seconds to read data for: "
        datatimes$ = str(datatime, nl$)
        store #$, datatimes$
        store #$, "Number of counts to move near wall: "
        rs$ = str(r, nl$)
        store #$, rs$
        store #$, "Muliplier once off wall: "
        mults$ = str(#7.3F, m!, nl$)
        store #$, mults$
        store #$, "Max step increment, in encoder counts: "
        maxcounts$ = str(maxcount, nl$)
        store #$, maxcounts$
        store #$, "Number of profiles: "
        npross$ = str(npros, nl$)
        store #$, npross$
        store #$, "Maximum total encoder counts (12800 per inch): "
        maxtotals$ = str(maxtotal, nl$)
```

```
        store #$, maxtotals$
return
specificnotes:
        answer4$ = "N"
        while(answer4$ <> "Y")
                // 1 byte = 1 character
                input "Enter Additional Notes (limit 80 characters): " note$, #80
                print "You entered: ", note$
                input "Is this correct <Y or N>: " answer4$, #1
        wend
        // while loop terminates when user enters Y
        store #$, "Notes: ", note$, nl$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0
return
do_over:
        print "Input Y to repeat the just-completed task exactly"
      print "Any other command will return you back to the previous menu:"
        input answer2$, #1
return
////////////////// STARTUP SUBROUTINE ///////////////////////
startup:
        pclr 16, 17, 18, 19, 20, 21, 22, 23     // Make sure no power is applied to all main
board/satellite outputs/ all logic 0 (false)
        pclr 1, 3, 5, 7                         // Power off and logic 0 (false) for all CVA-
related outputs
        cbreak quick_exit                       // goto quick_exit label when Ctrl-C pressed
        error = 0
        onerr quick_exit, error                 // goto quick_exit if execution error occurs,
                                                        // error information stored in
'error'
        lowbatvalue! = 5.3                      // Initialize low-battery threshhold
        nl$ = str(\13,\10)                      // a new-line string for formatting output
        if (DFERASED = 0)
                print nl$, "WARNING: data file not erased"
                print "Enter Y to continue, otherwise program will terminate"
                input answer$,#1
                if (answer$ <> "Y")
                        stop
                endif
        endif
return
////////////////////////////////////////////////////////
/////////////// QUICK_EXIT SUBROUTINE ///////////////////
// Control-C was pressed, OR error occured, exit gracefully
// Error List on page 144 of TFBasic Manual
```

```
quick_exit:
        pclr 16, 17, 18, 19, 20, 21, 22, 23     // Make sure no power is applied to all main
board/satellite outputs/ all logic 0 (false)
        pclr 1, 3, 5, 7                         // Power off and logic 0 (false) for all CVA-
related outputs
        count                                   // Make sure background counter is off
        call &hFD88,0                           // flush output buffer to datafile
        if (error = 0)                          // error did not occur
                print "Control-C Pressed, Program Stopped, Ready for Data Offload."
        else                                    // error did occur
                message$ = str("Error #", error / 65536, "  @", #05H, error % 65536)
                print message$
                store #$, message$, nl$
                call &hFD88,0                   // flush output buffer to datafile
        endif
        stop
return                                          // Program will never reach this return
statement
////////////////////////////////////////////////////
//////////////////BATT_CHECK SUBROUTINE //////////////////
batt_check:
        pset 17                                 // turn on power to main sensors & batt
check circuit
        sleep 0                                 // slight warm up delay
        sleep 20
        batv! = 0
        batvave! = 0
        batvf! = 0
        for k = 1 to 50                         // 50 samples to be sure
                sleep 0
                sleep 5
                kk! = k
                batv! = chan(3)
                batvave! = batvave! + ((batv! - batvave!) / kk!)
        next k
        pclr 17                                         // turn off power to main sensors
        batvf! = (batvave! * 3 * .0000763)              // 1/3 divider circuit in -AV3
        if (batvf! < lowbatvalue!)
                message$ = str("BATTERY < ", #.3F, lowbatvalue!, " V, SHUTTING
DOWN, V = ", #.3F, batvf!)
                store #$, message$, nl$
                print nl$, message$
                call &hFD88,0                   // flush data buffer
                gosub stage_down                // move stage down
                gosub shutdown
        endif
```

```
return
////////////////////////////////////////////////////////
/////////////// INPUT_TIME SUBROUTINE ////////////////////
input_time:
        answer$ = "N"
        while (answer$ <> "Y")                          // re-input time until user enters "Y"
                print
                input "Year:   "?(5)
                input "Month:  "?(4)
                input "Day:    "?(3)
                input "Hour:   "?(2)
                input "Minute: "?(1)
                input "Second: "?(0)
                stime
                setrtc
                times$ = str(#02, ?(4),"/",?(3),"/",?(5)," ",?(2),":",?(1),":",?(0))
                print "The time stamp is: ", times$
                print "Does this data look correct?"
                print "Input Y to continue, any other command will allow you to redo
time."
                input answer$, #1
        wend
return
////////////////////////////////////////////////////////
/////////////// NEXT_STAGE_PT SUBROUTINE ///////////////
next_stage_pt:
        badmove = 0                     // badmove gets set to 1 if stage fails to move
        oldtar = target
        if (iypt > 8)     // the wall is 8 ypts long
                target = oldtar * m!
        endif
        if (target > maxcount)
                target = maxcount
        endif
        pset 18                         //turn ON stage encoder power
        sleep 0
        sleep 20
        // turn on background counting process to count encoder pulses
        backcount = 0
        count backcount
        readrtc // copy PIC time to ?-variable
        starttime = ?
// PWM parameters--added 1/30/2013 for v2 to allow smaller steps
// on_time is PWM Duty Cycle as an integer percentage"
// where 10 = 10% and 100 = 100%"
        on_time =  7
```

```
        off_time = 100 - on_time
        pclr 19 // set direction UP
//      pset 16  // turn ON stage motor power
        while (backcount < target)
// PWM loop ...
                pset 16 // stage pwr on
                for i = 1 to on_time
                next i
                pclr 16 // stage pwr off
                for i = 1 to off_time
                next i
        wend
        pclr 16                 // turn OFF stage motor power
        sleep 0                 // give stage time to come to a stop
        sleep 50
        count                   // turn off background counting
        pclr 18                 // turn OFF encoder power to limit battery power drain
        // done with stage move, update encoder total counts
        totalcount = totalcount + backcount
        // done with stage move, now at next ypt
        iypt = iypt + 1
return
////////////////////////////////////////////////////
/////////////// LOCHK SUBROUTINE ///////////////////////
// Ensure stage is at lower limit to start profile
// Check to see if stage is at lower limit, if not,
// try to drive twice more at most
// lower limit reached if PA4=0 when RB3=1
// upper limit reached if PA2=0 when RB3=0
lochk:
        pset 19
        lolim = pin(4)
        pclr 19
        attempt = 1
        while (lolim <> 0 & attempt <= 8)
                message$ = str("lochk: Low limit not reached, lolim = ", lolim)
                print message$                  // for benchtop testing
                store #$, message$, nl$         // store message to datafile
                call &hFD88,0                   // flush datafile buffer
                gosub stage_down                // redrive stage
                pset(19)
                lolim = pin(4)                  // low limit reached if lolim = 0
                pclr(19)
                attempt = attempt + 1           // increment number of redrive attempts
        wend
return
```

```
////////////////////////////////////////////////////
//////////////// PROFILE_HEADER SUBROUTINE ////////////////
profile_header:
        //store and print header data for current profile
        readrtc
        rtime
        times$ = str(#02, ?(5),"/",?(4),"/",?(3)," ",?(2),":",?(1),":",?(0), ",")
        store #$, times$
        ipros$ = str(" profile no, ", ipro, ", ")
        store #$, ipros$
        lolims$ = str("lolim if 0, ", lolim, nl$)
        store #$, lolims$
        store #$, "Time,MPitot(V),Static(V),Preston(V),Temperature(K),Voltage,Encoder
totcnt", nl$
        print nl$, times$, ipros$, lolims$ ;
        print nl$,
"Time,MPitot(V),Static(V),Preston(V),Temperature(K),Voltage,Encoder totcnt", nl$
return
////////////////////////////////////////////////////
//////////////// STAGE_DOWN SUBROUTINE ////////////////
stage_down:
        // move stage DOWN all the way
        pset 19
        pset 16
        sleep 0
        sleep 1000
        pclr 16
        pclr 19
return
////////////////////////////////////////////////////
//////////////// SHUTDOWN SUBROUTINE ////////////////
shutdown:
        //CLEAR OUTPUTS, just to be sure
        pclr 16, 17, 18, 19, 20, 21, 22, 23     // Make sure no power is applied to all main
board/satellite outputs/ all logic 0 (false)
        pclr 1, 3, 5, 7                         // Power off and logic 0 (false) for all CVA-
related outputs
        stop                                    // go ahead and stop, power-on but in lo-
power config
return
////////////////////////////////////////////////////
//////////////// TAKE_PT_DATA SUBROUTINE ////////////////
take_pt_data:
        //DATA ACQUISITION
        //pause 5 seconds after move
        sleep 0
```

```
sleep 500
//turn ON power to main sensors and slight warm-up delay
pset 17
sleep 0
sleep 20
//read & average main sensor
pitotsignal! = 0
pitotave! = 0
staticsignal! = 0
staticave! = 0
prestonsignal! = 0
prestonave! = 0
Te! = 0
tempave! = 0
batv! = 0
batvave! = 0
for k = 1 to iterations
        sleep 0
        sleep sleeptime
        kk! = k
        pitotsignal! = chan(7)
        pitotave! = pitotave! + ((pitotsignal! - pitotave!) / kk!)
        staticsignal! = chan(5)
        staticave! = staticave! + ((staticsignal! - staticave!) / kk!)
        prestonsignal! = chan(9)
        prestonave! = prestonave! + ((prestonsignal! - prestonave!) / kk!)
        Te! = chan(0)
        tempave! = tempave! + ((Te! - tempave!) / kk!)
        batv! = chan(3)
        batvave! = batvave! + ((batv! - batvave!) / kk!)
next k
//turn off power to main sensors
pclr 17
//compute sensor avg voltages, battery avg voltages, & temp in C
pitot! = (pitotave! * .0000763)
static! = (staticave! * .0000763)
preston! = (prestonave! * .0000763)
T! = (tempave! * .00763) - 273.0            // 100.0 * (tempave! * 0.0000763) -
273.0
batvf! = (batvave! * 3 * .0000763)          // 1/3 divider circuit in -AV3
readrtc
rtime
//DATA STORAGE
times$ = str(#02, ?(5),"/",?(4),"/",?(3)," ",?(2),":",?(1),":",?(0), ",")
store #$, times$
pitots$ = str(#7.3F, pitot!, ",")
```

```
        store #$, pitots$
        statics$ = str(#7.3F, static!, ",")
        store #$, statics$
        prestons$ = str(#7.3F, preston!, ",")
        store #$, prestons$
        temps$ = str(#7.2F, T!, ",")
        store #$, temps$
        batvfs$ = str(#7.3F, batvf!, ",")
        store #$, batvfs$
        encts$ = str(" ", totalcount, ",", nl$)
        store #$, encts$
        //call to &hFD88 ensures that data buffer is flushed
        call &hFD88,0
        //PRINT TO SCREEN (for benchtop monitor/checkout)
        print times$, pitots$, statics$, prestons$, temps$, batvfs$, encts$, nl$
return
////////////////////////////////////////////////////////
```

AMETEK Rotron 90-AA3-126

- 90-AA3-126

MADE IN THE USA

Part Number	018085000
Unit Description	90-AA3-126
Airflow (CFM)	275.46
Pressure Shutoff (IWG)	5.75
Line Voltage (DC)	28
Phase/Frequency (Hz)	N/A
Capacitor (µF)	
Power (Watts)	0
Run Current (Amps)	10
Locked Rotor (Amps)	0
RPM	24000
Weight (lbs/kg)	0 / 1
DBA at 3 feet	0
Max Ambient Temp (°C)	70

Life Expectancy

Contact AMETEK Rotron for specific ratings. Rotron rates life expectancy in hours L10 or MTBF at a given ambient temperature. This is the time at which 90%, for L10, and 50%, for MTBF, of an available test lot are still operating.

Accessories:

Low Speed Warning Detector
Air Flow Switch
Finger Guard

Environmental Resistance Guide

Designed to meet MIL-STD-810C

Altitude	Method 500.1, pr I
Acceleration	Method 513.2
Vibration	Method 514.2
Shock	Method 516.2
Salt Fog	Method 509.1
Rain	Method 506.1
Fungus	Method 508.1
Humidity	Method 507.1, pr I, II, III
Explos. Atm.	Method 511.1, pr I

For specifics, please contact the factory.

Woodstock, N.Y. USA (P) 845-679-2401 (F) - 845-679-1870 (email) milinquiry@ametek.com

AMETEK®
ROTRON TECHNICAL MOTOR DIVISION

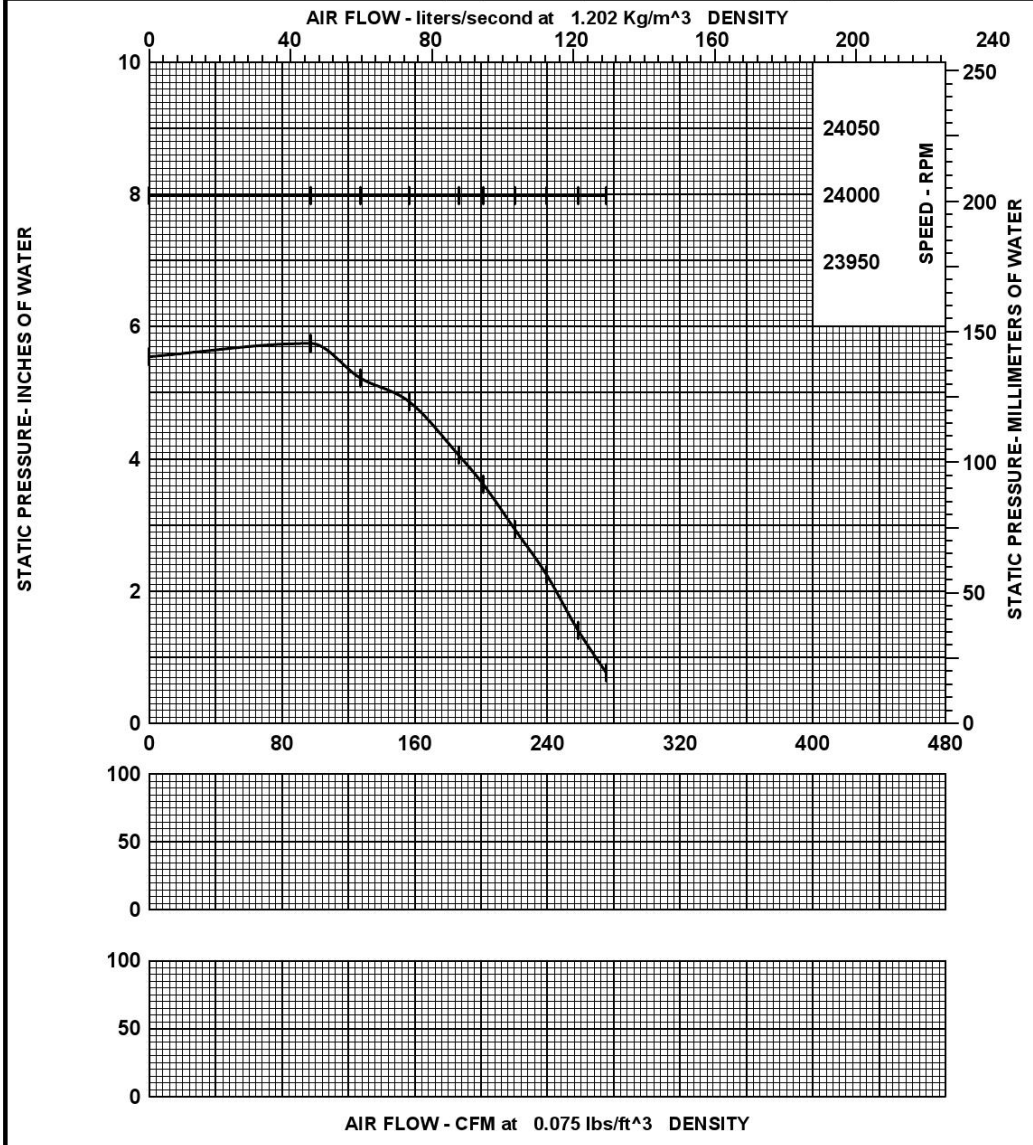
Measured:	Project:	Test:	Model:	Axial
S.Mogan	Typical Curve	AS0019-2	Type:	90-AA3-126

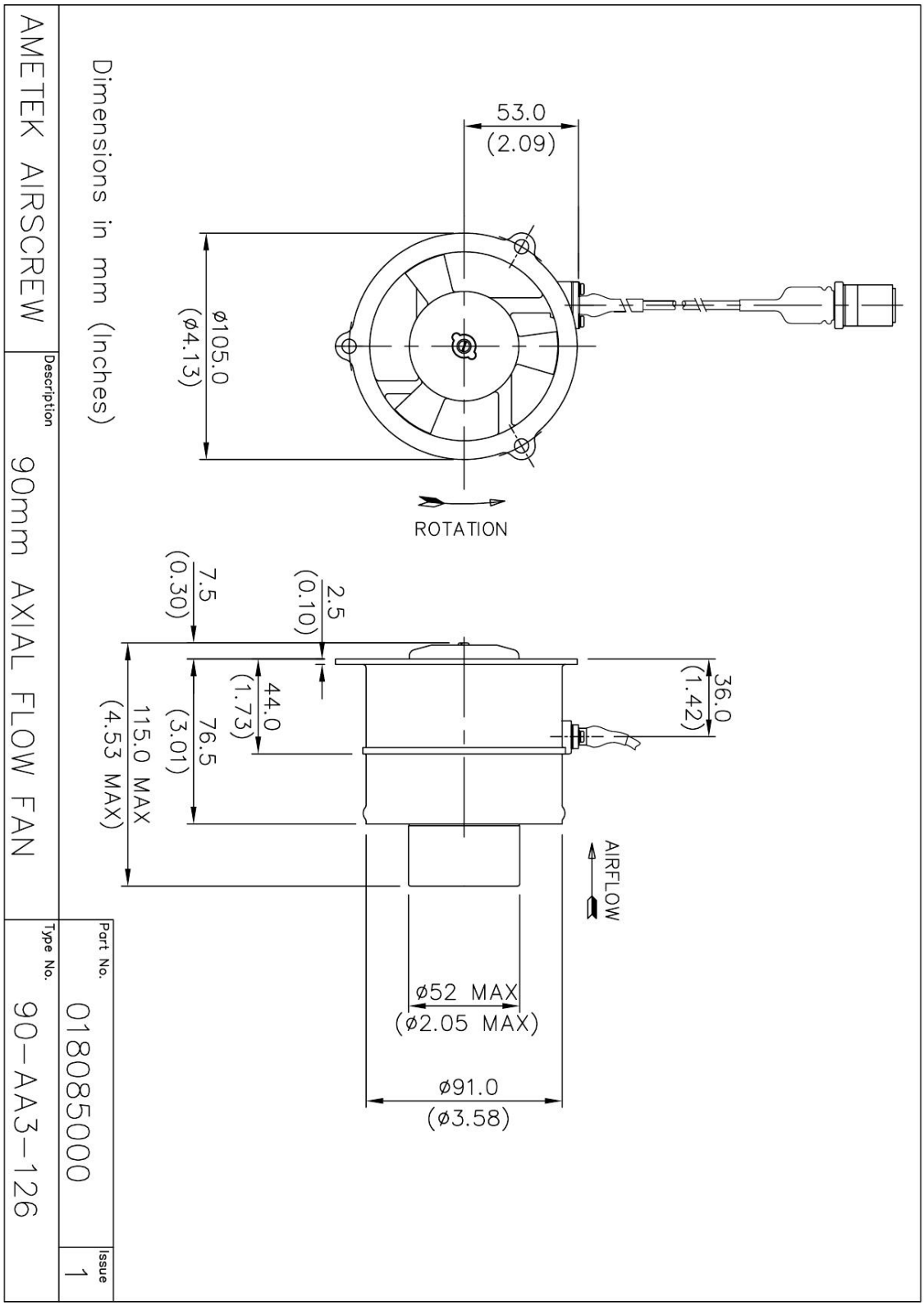
NOTES:

Date:	1-Oct-08	
Bench:	Other...	
Flow:	L ☐ R ☐	

FREE FLOW		
RPM	24000	
Line Amps		
Locked Amps	---	
Watts		

MOTOR	
Series	0
Type	DC
Volts	28
Wave	DC
Phase	DC
Hertz	NA
MFD	NA




AMETEK AIRSCREW
Description
90mm AXIAL FLOW FAN
Type No.
90-AA3-126
Part No.
018085000
Issue
1
Dimensions in mm (Inches)
53.0
(2.09)
ø105.0
(ø4.13)
ROTATION
7.5
(0.30)
2.5
(0.10)
44.0
(1.73)
76.5
(3.01)
115.0 MAX
(4.53 MAX)
36.0
(1.42)
AIRFLOW
ø52 MAX
(ø2.05 MAX)
ø91.0
(ø3.58)